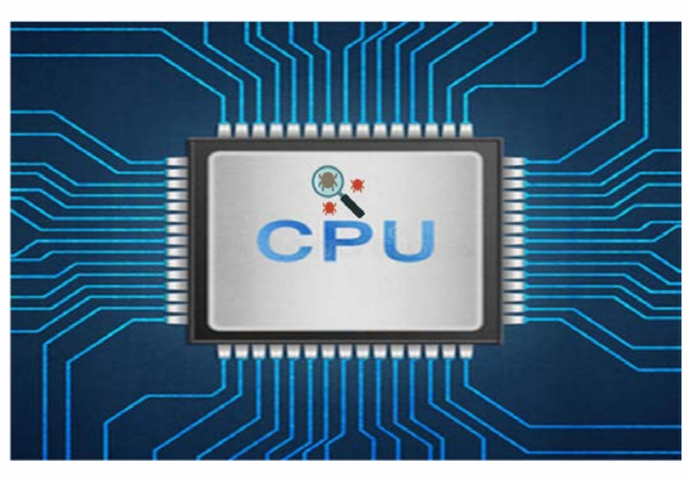
ISCAS 中国科学院软件研究所学术年会'2024
暨 重 点 实 验 室 科 技 活 动 周
学术论文

SEPE-SQED: Symbolic Quick Error Detection by Semantically Equivalent Program Execution
基于语义等价程序执行的符号化快速错误检测

（DAC' 2024录用）



[1,2]李宇锋，[2]杨秋松*，[2]慈轶为，[1,2]田恩源

[1]中国科学院大学　[2]中国科学院软件研究所基础软件国家工程研究中心
crazybinary494@gmail.com, {qiusong, yiwei}@iscas.ac.cn, tianenyuan@nfs.iscas.ac.cn

1. 介绍

处理器验证非常困难

- 现代处理器实现涉及复杂的微架构优化（分支预测、乱序执行、预取策略）
- 模拟(simulation)/仿真(emulation)手段覆盖不到系统的所有出错行为，容易遗漏边角情况（corner cases）
- 形式化验证（formal verification）建立待验证设计的数学模型，通过逻辑推理的方式证明系统是否满足期望属性，擅长发现边角错误。但验证的属性需要手工写出，面临以下挑战：
  - ◆编写过程要求验证人员具备丰富的形式化验证经验，还要求精通微架构设计细节
  - ◆耗时长，容易出错和遗漏
- 验证工作量在项目中的平均占比超过50%，但是仅有24%的项目能够实现首次流片成功[1]

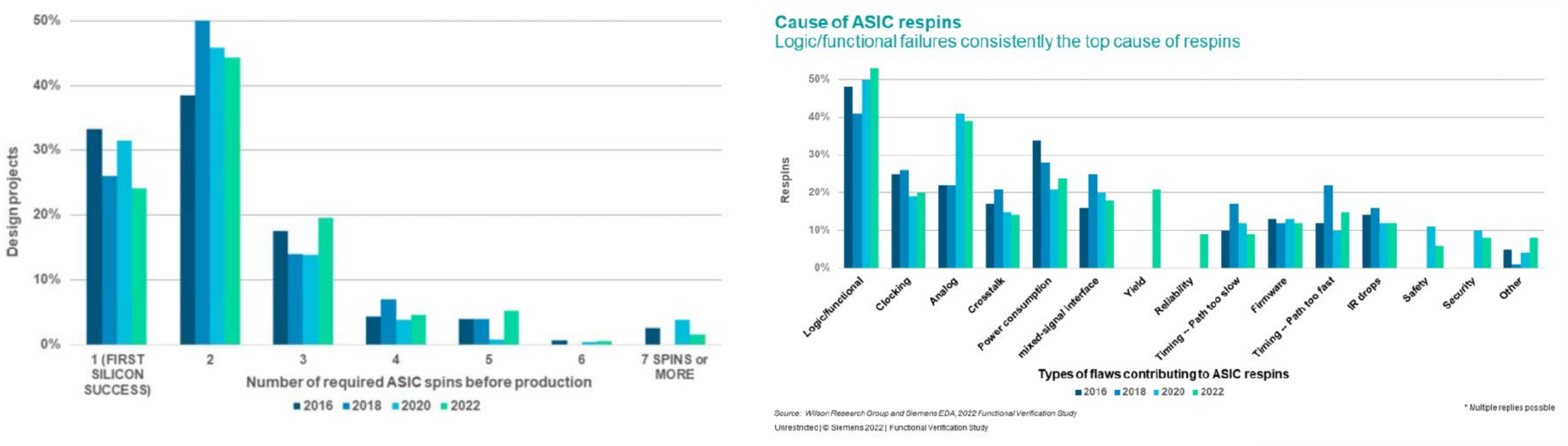
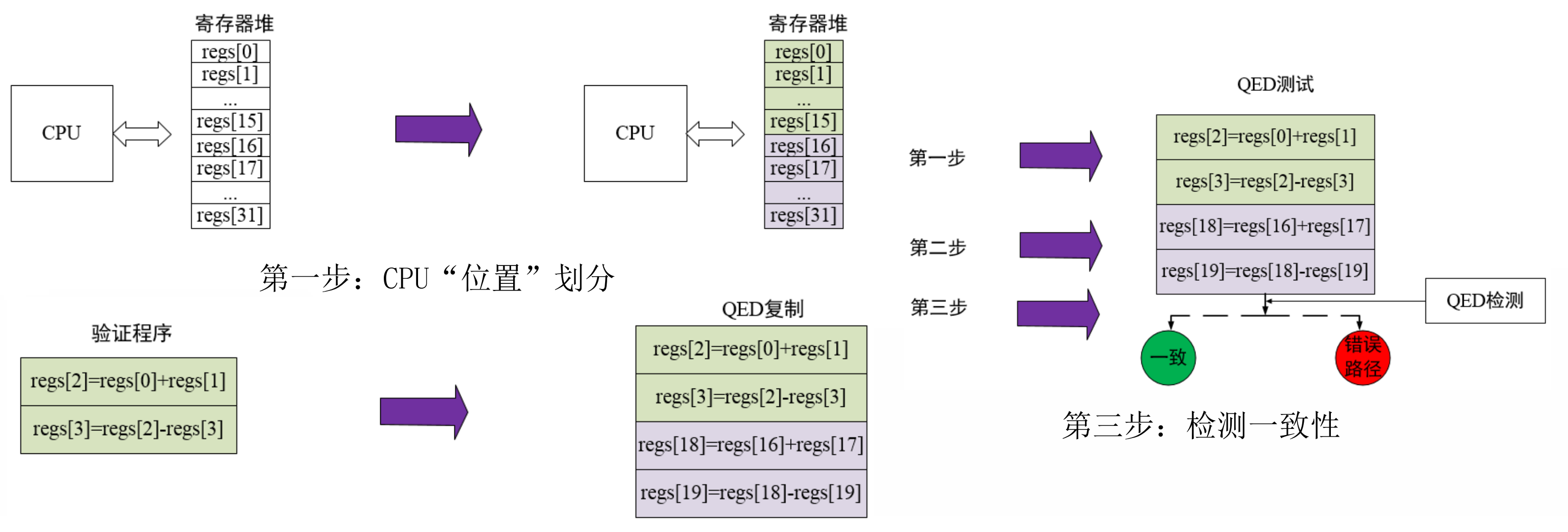


图1：威尔逊研究集团调研报告

2. 处理器模型检测领域的变革：符号化快速错误检测（Symbolic Quick Error Detection, SQED[2,3,4,5,6]）

- 全自动
  - ◆只验证一个通用属性——自一致性（self-consistency）
    - ✓实现正确的处理器执行一段原始指令和一段与原始指令功能一样的复制指令须得到一致的架构状态
  - ◆与微架构设计无关
  - ◆只需少量手工操作和少量形式化验证专业知识
- 鲁棒
  - ◆符号化地系统枚举长度递增的指令序列
- 快速
  - ◆找到从初始状态开始的最短错误轨迹
- 有效（对比传统方法）
  - ◆检测时间缩短100倍
  - ◆错误轨迹缩短10^6倍

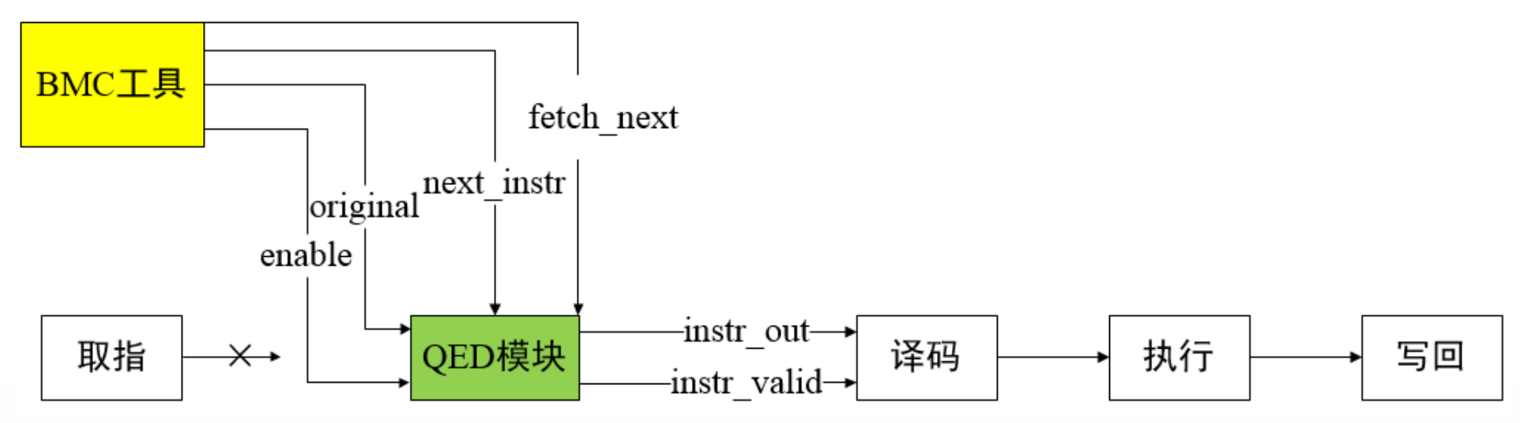


第一步：CPU“位置”划分

第二步：程序复制

第三步：检测一致性

图2：QED测试



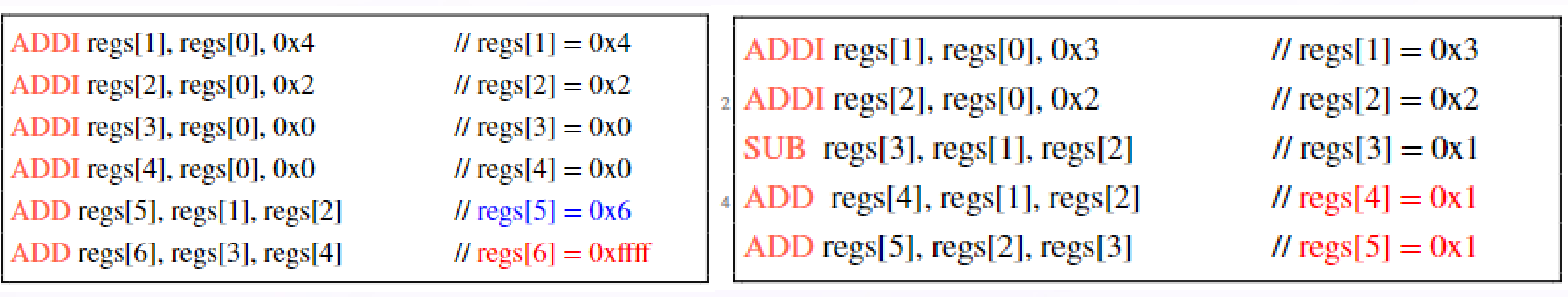
SQED：QED+BMC(Bounded Model Checking)

图3：SQED

3. 当前工作的不足

处理器中的逻辑错误可以分成两类

- 单指令类型的错误（single-instruction bugs）
  - ◆ 与程序环境无关
  - ◆ 激活错误然后将错误传播到架构可见状态的是同一条指令



```
ADDI regs[1], regs[0], 0x4      // regs[1] = 0x4
ADDI regs[2], regs[0], 0x2      // regs[2] = 0x2
ADDI regs[3], regs[0], 0x0      // regs[3] = 0x0
ADDI regs[4], regs[0], 0x0      // regs[4] = 0x0
ADD regs[5], regs[1], regs[2]   // regs[5] = 0x6
ADD regs[6], regs[3], regs[4]   // regs[6] = 0xffff
```

```
ADDI regs[1], regs[0], 0x3      // regs[1] = 0x3
ADDI regs[2], regs[0], 0x2      // regs[2] = 0x2
SUB  regs[3], regs[1], regs[2]  // regs[3] = 0x1
ADD  regs[4], regs[1], regs[2]  // regs[4] = 0x1
ADD regs[5], regs[2], regs[3]   // regs[5] = 0x1
```

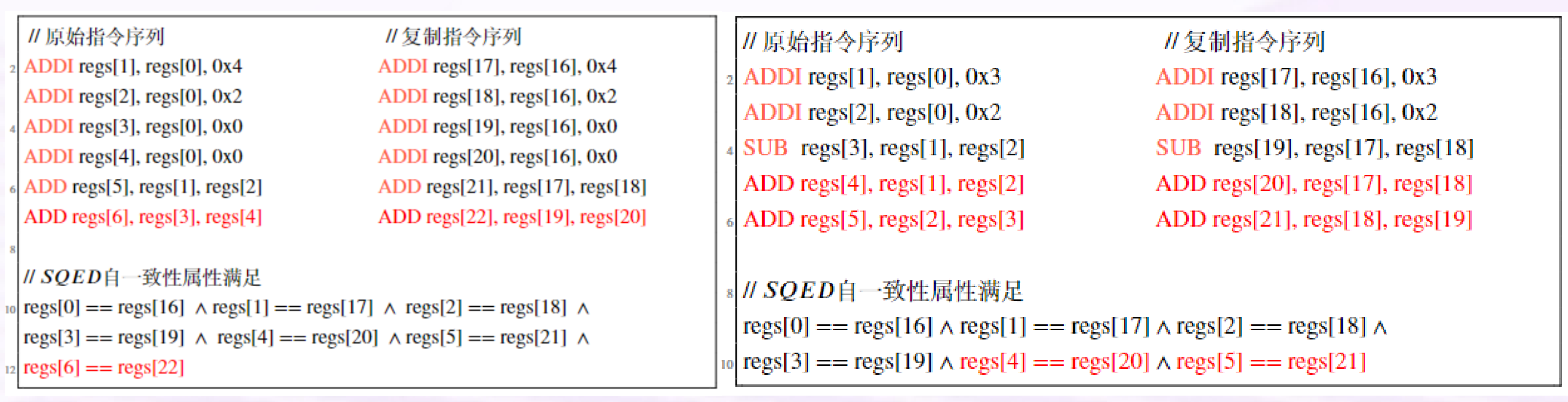
图4：单指令类型的错误

- 多指令类型的错误（multiple-instruction bugs）
  - ◆ 与程序环境相关
  - ◆ 一串特定的指令序列激活该错误，然后一条指令将错误的效果传播到架构可见状态

```
MULH regs[3], regs[1], regs[2]  // 激活错误
MULH regs[6], regs[4], regs[5]  // 错误发生：MULH 指令的第一个源操作数损坏
NOP                             // 不激活错误
NOP                             // 不激活错误
MULH regs[6], regs[4], regs[5]  // MULH 指令执行正确
```

图5：多指令类型的错误

- SQED漏检单指令类型的错误！！！
  - ◆ 单指令类型错误会同效果地影响到原始指令和复制指令的执行结果
    - ✓ 原始寄存器和复制寄存器的值总保持一致，自一致性属性不会被违反，验证结果“假阳性”

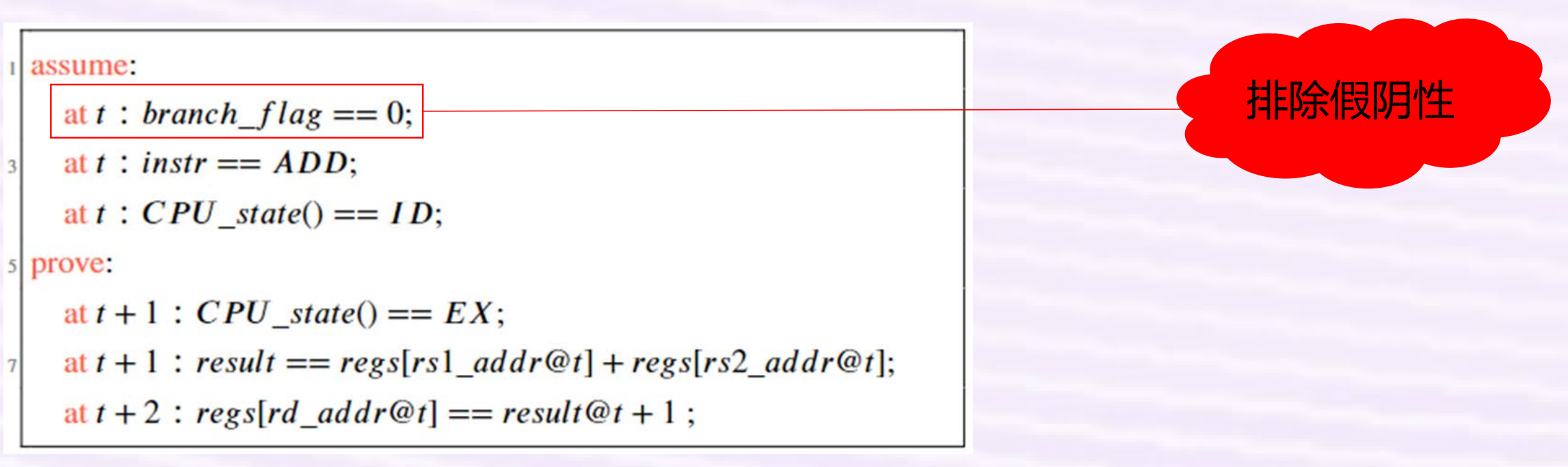


```
// 原始指令序列                     // 复制指令序列
ADDI regs[1], regs[0], 0x4         ADDI regs[17], regs[16], 0x4
ADDI regs[2], regs[0], 0x2         ADDI regs[18], regs[16], 0x2
ADDI regs[3], regs[0], 0x0         ADDI regs[19], regs[16], 0x0
ADDI regs[4], regs[0], 0x0         ADDI regs[20], regs[16], 0x0
ADD regs[5], regs[1], regs[2]      ADD regs[21], regs[17], regs[18]
ADD regs[6], regs[3], regs[4]      ADD regs[22], regs[19], regs[20]

// SQED自一致性属性满足
regs[0] == regs[16] ∧ regs[1] == regs[17] ∧ regs[2] == regs[18] ∧
regs[3] == regs[19] ∧ regs[4] == regs[20] ∧ regs[5] == regs[21] ∧
regs[6] == regs[22]
```

```
// 原始指令序列                     // 复制指令序列
ADDI regs[1], regs[0], 0x3         ADDI regs[17], regs[16], 0x3
ADDI regs[2], regs[0], 0x2         ADDI regs[18], regs[16], 0x2
SUB  regs[3], regs[1], regs[2]     SUB  regs[19], regs[17], regs[18]
ADD regs[4], regs[1], regs[2]      ADD regs[20], regs[17], regs[18]
ADD regs[5], regs[2], regs[3]      ADD regs[21], regs[18], regs[19]

// SQED自一致性属性满足
regs[0] == regs[16] ∧ regs[1] == regs[17] ∧ regs[2] == regs[18] ∧
regs[3] == regs[19] ∧ regs[4] == regs[20] ∧ regs[5] == regs[21]
```

图6：单指令类型的错误逃逸SQED检测

- ◆ 现有的弥补措施效率不高
  - ✓ 由于每条指令操作数的范围太大，模拟/仿真的测试用例有限
  - ✓ 构造验证单条指令规范的属性脱离不了微架构设计细节

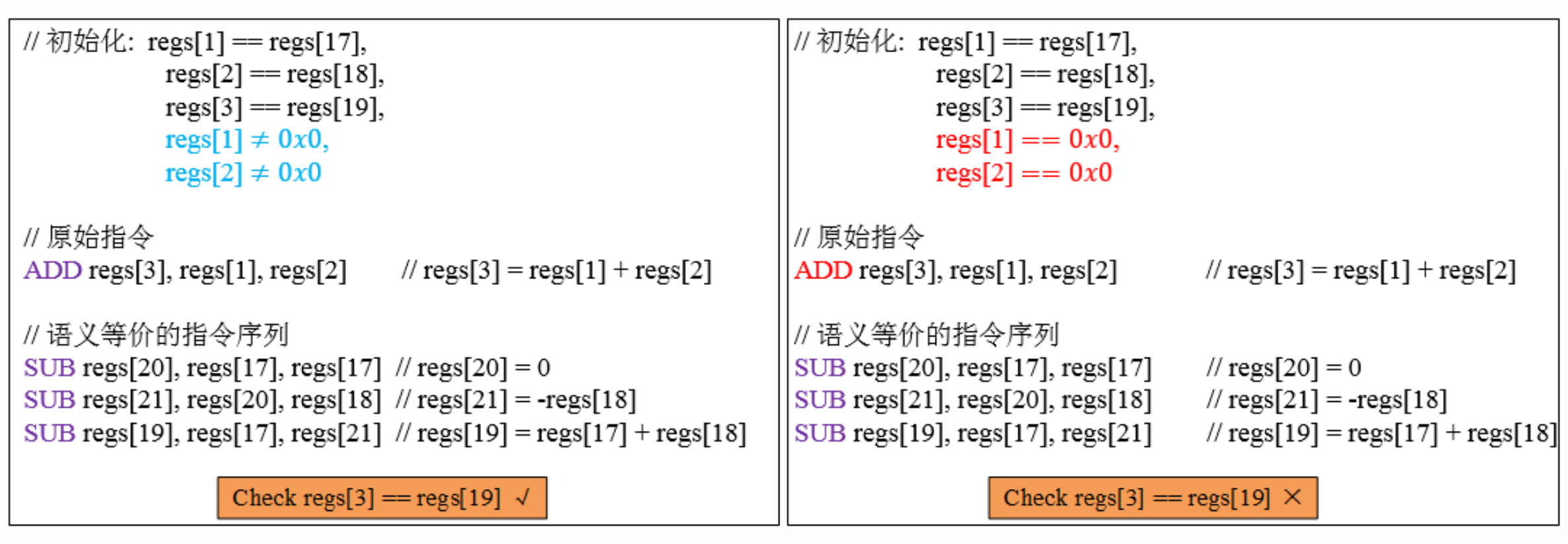


```
assume:
    at t : branch_flag == 0;
    at t : instr == ADD;
    at t : CPU_state() == ID;
prove:
    at t+1 : CPU_state() == EX;
    at t+1 : result == regs[rs1_addr@t] + regs[rs2_addr@t];
    at t+2 : regs[rd_addr@t] == result@t+1;
```

图7：描述ADD指令规范的非通用属性

4. 贡献

- 泛化自一致性通用属性
  - ◆ 单指令类型的错误不会同效果地影响到原始指令和其语义等价的指令序列



```
// 初始化: regs[1] == regs[17],
           regs[2] == regs[18],
           regs[3] == regs[19],
           regs[1] ≠ 0x0,
           regs[2] ≠ 0x0

// 原始指令
ADD regs[3], regs[1], regs[2]        // regs[3] = regs[1] + regs[2]

// 语义等价的指令序列
SUB regs[20], regs[17], regs[17]     // regs[20] = 0
SUB regs[21], regs[20], regs[18]     // regs[21] = -regs[18]
SUB regs[19], regs[17], regs[21]     // regs[19] = regs[17] + regs[18]

Check regs[3] == regs[19] √
```

```
// 初始化: regs[1] == regs[17],
           regs[2] == regs[18],
           regs[3] == regs[19],
           regs[1] == 0x0,
           regs[2] == 0x0

// 原始指令
ADD regs[3], regs[1], regs[2]        // regs[3] = regs[1] + regs[2]

// 语义等价的指令序列
SUB regs[20], regs[17], regs[17]     // regs[20] = 0
SUB regs[21], regs[20], regs[18]     // regs[21] = -regs[18]
SUB regs[19], regs[17], regs[21]     // regs[19] = regs[17] + regs[18]

Check regs[3] == regs[19] ×
```

图8：ADD指令和三条SUB指令表达相同语义

- ◆ 实现正确的处理器执行原始指令与执行跟原始指令语义等价的指令序列须得到一致的架构状态——广义自一致性通用属性
- ◆ SEPE-SQED：验证广义自一致性通用属性的符号化快速错误检测

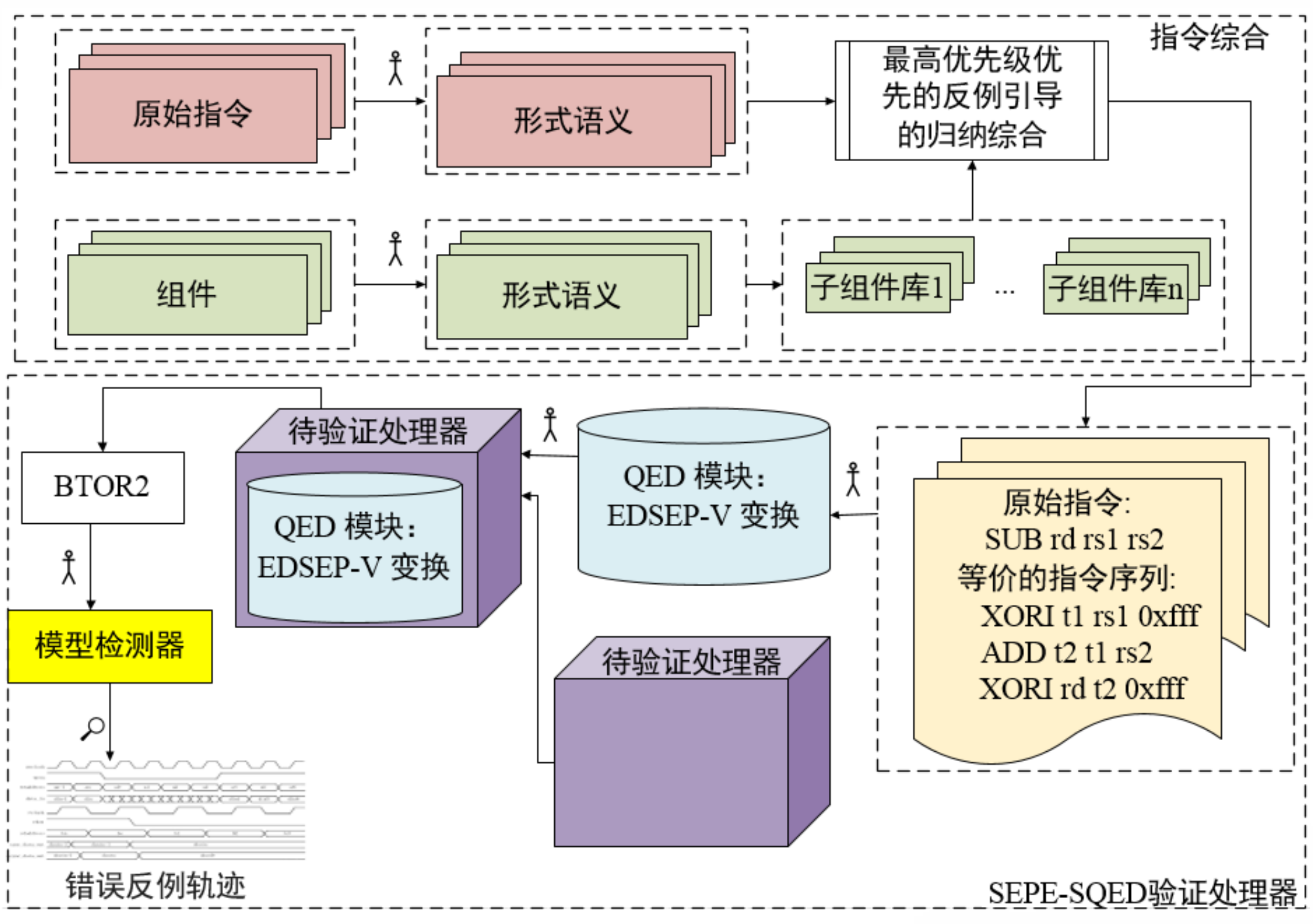


图9：SEPE-SQED

- ✓ 第一步：利用程序综合算法寻找每条原始指令的语义等价的指令序列
  - ➢ 结果作为QED测试中原始指令的变换规则
- ✓ 第二步：利用第一步得到的变换规则做符号化快速错误检测时的指令变换，检测处理器执行完原始指令与其语义等价指令序列之后的架构状态是否一致。

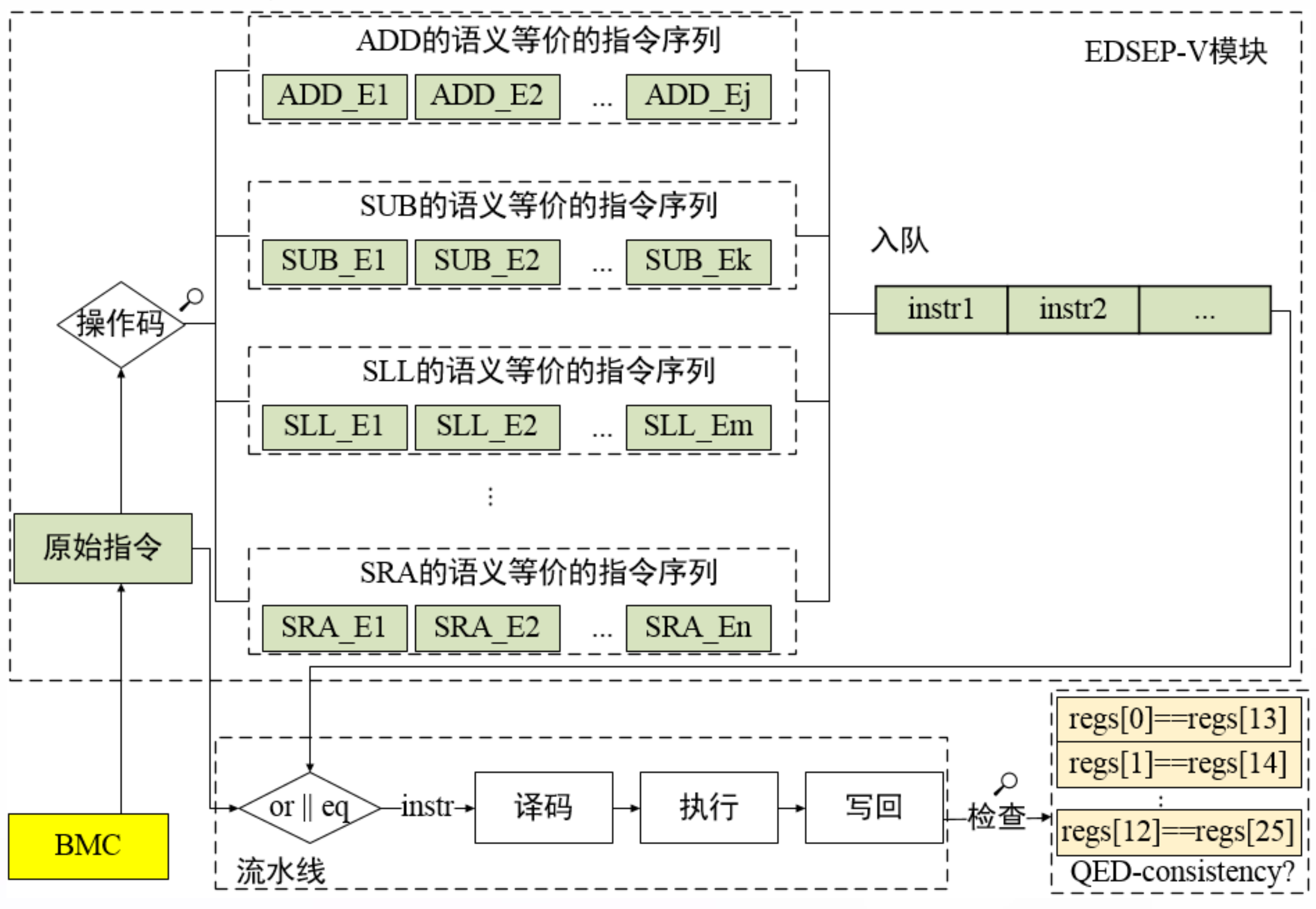


图10：验证过程

- 指令综合（$\exists P: \forall \vec{I}, O: (P(\vec{I}) == O) \Rightarrow (instr_g(\vec{I}) == O)$）

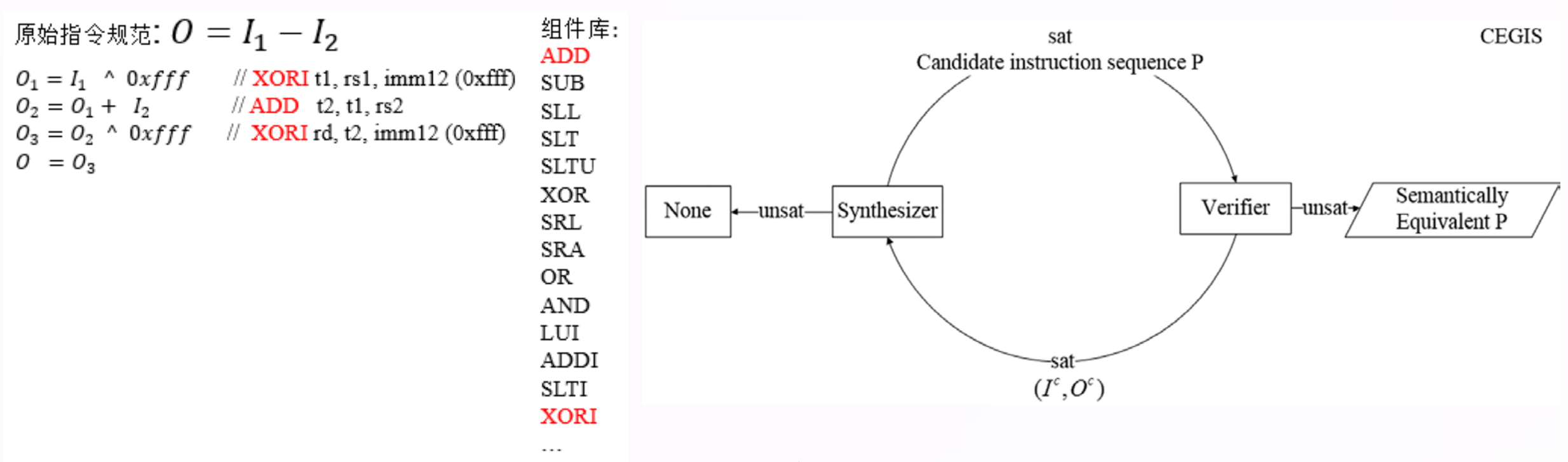


图11：指令综合过程

- ◆ 组件的数目过多会严重影响综合的效率
  - ✓ 组件组合的数量随着组件数目非线性增长
  - ✓ 很多无效的综合过程浪费时间
- ◆ 启发式综合算法（Counterexample-Guided Inductive Synthesis based on the Highest Priority First, HPF-CEGIS）
  - ✓ 每个组件赋予一对优先权重
    - ➢ 选择权重越大组件被选择的概率越大
    - ➢ 排它权重越大组件不被选择的概率越大
  - ✓ 动态反馈机制调节组件权重，筛选出当前最有可能综合出原始指令的组件组合

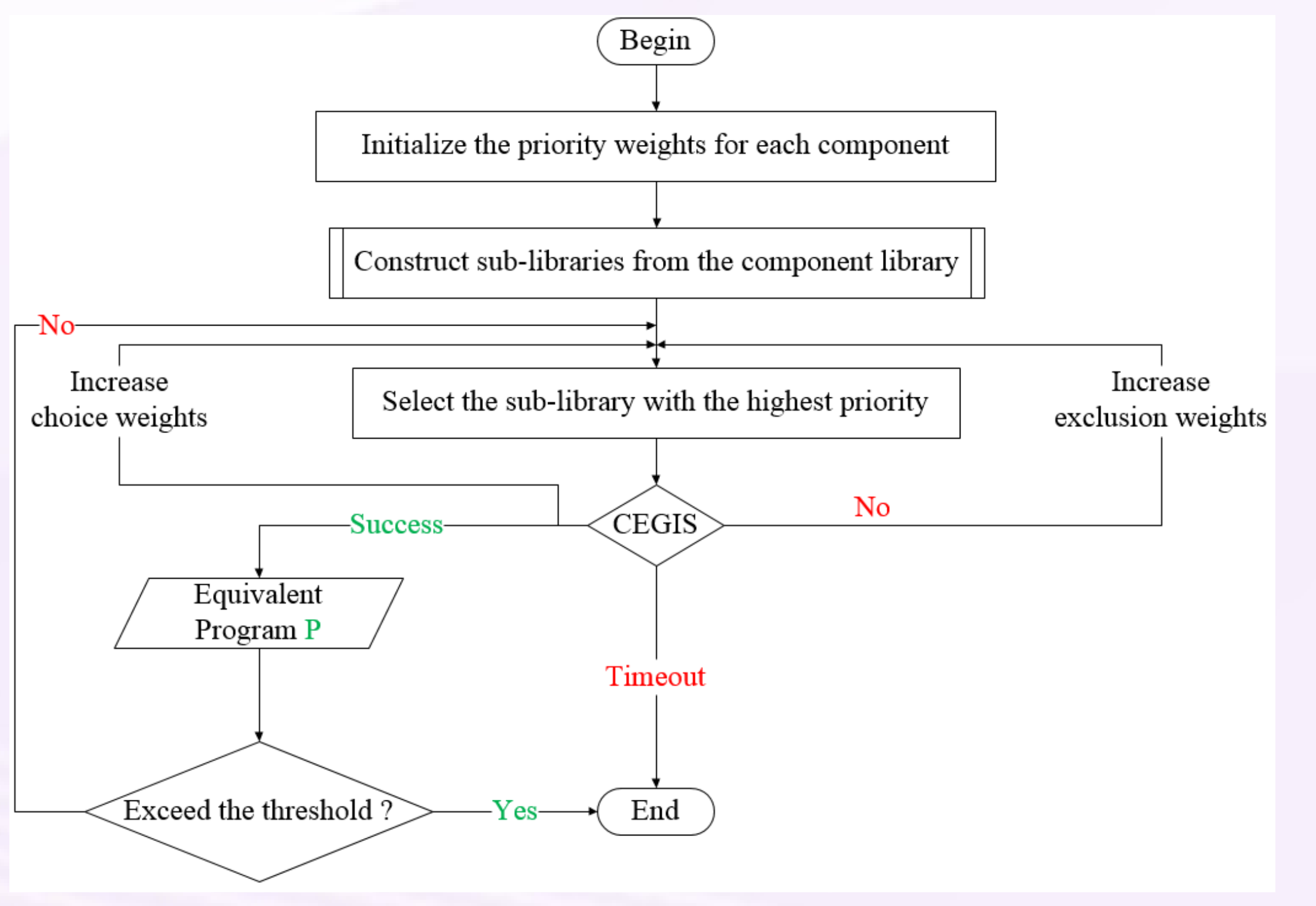
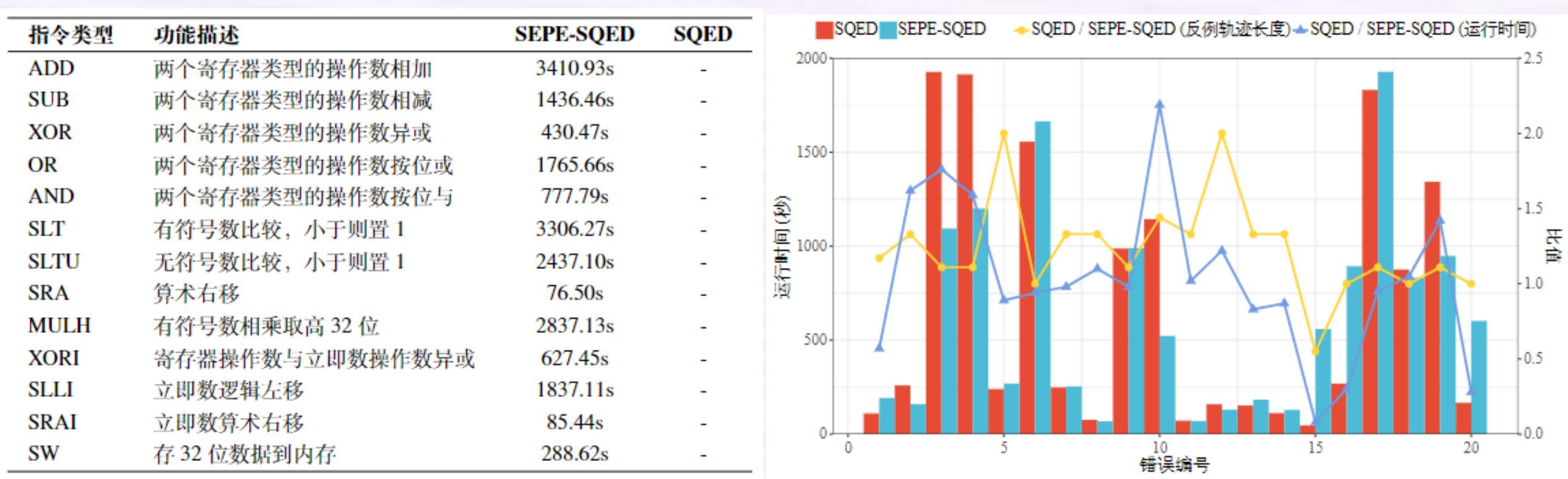


图12：HPF-CEGIS

5. 实验评估

- SEPE-SQED成功检测到突变测试的所有逻辑错误（单指令类型、多指令类型）
- 对于某些错误，找到比SQED更短的错误轨迹



指令类型	功能描述	SEPE-SQED	SQED
ADD	两个寄存器类型的操作数相加	3410.93s	-
SUB	两个寄存器类型的操作数相减	1436.46s	-
XOR	两个寄存器类型的操作数异或	430.47s	-
OR	两个寄存器类型的操作数按位或	1765.66s	-
AND	两个寄存器类型的操作数按位与	777.79s	-
SLT	有符号数比较，小于则置 1	3306.27s	-
SLTU	无符号数比较，小于则置 1	2437.10s	-
SRA	算术右移	76.50s	-
MULH	有符号数相乘取高 32 位	2837.13s	-
XORI	寄存器操作数与立即数操作数异或	627.45s	-
SLLI	立即数逻辑左移	1837.11s	-
SRAI	立即数算术右移	85.44s	-
SW	存 32 位数据到内存	288.62s	-

单指令类型错误的检测

多指令类型的错误检测

图13：实验结果

参考文献

[1] Research W. The 2022 wilson research group functional verification study [EB/OL]. https://blogs.sw.siemens.com/verificationhorizons/2022/10/10/prologue-the-2022-wilson-research-group-functional-verification-study/.

[2] Singh E, Lin D, Barrett C, et al. Symbolic quick error detection for pre-silicon and post-silicon validation: Frequently asked questions[J]. IEEE Design & Test, 2016, 33(6): 55-62.

[3] Singh E, Lin D, Barrett C, et al. Logic bug detection and localization using symbolic quick error detection[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[4] Singh E, Devarajegowda K, Simon S, et al. Symbolic qed pre-silicon verification for automotive microcontroller cores: Industrial case study[C]//2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019: 1000-1005.

[5] Lonsing F, Ganesan K, Mann M, et al. Unlocking the power of formal hardware verification with cosa and symbolic qed[C]//2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2019: 1-8.

[6] Lonsing F, Mitra S, Barrett C. A theoretical framework for symbolic quick error detection[C]//# PLACEHOLDER_PARENT_METADATA_VALUE#. TU Wien Academic Press, 2020, 1: 26-35.