本文为对江南大学队伍 aisystem(队伍编号 T202510295205602)，在 2025 年全国大学生计算机系统能力大赛-编译系统设计赛-编译系统挑战赛代码体积赛道中的**严重违规行为**说明。

该队伍的违规行为总结如下：

一、通过识别测试用例文件名的方式，对测试用例进行针对性优化。
二、对于测试用例名，采用加密混淆的方式试图掩盖作弊事实。
三、针对每个用例，人工标记用例中的符号名进行删除。在没有输出的测试用例 bm3 上，进行了删除程序正常功能的优化，用无功能的文件伪造用例结果。
四、对未公开的决赛用例，通过某种方式标记识别其文件名与符号名，进行针对性优化。
五、通过硬编码调整偏移量，进行针对测评系统的优化，在其他环境会报错。

以下是对于其开源项目的违规行为分析和相关证明材料。
aisystem 队伍仓库地址(已被删除)：
https://gitlab.eduxiji.net/T202510295205602/tiny
官方 fork 的 aisystem 队伍仓库地址：
https://gitlab.eduxiji.net/educg-group-36295-2824709/T202510295205602-63
为防止其修改删除，我们对其进行了备份，备份仓库地址：
https://github.com/hachimi-jn/2025-bisheng-cup-codesize-jiangnan-university-aisystem-tiny

一、竞赛简介

比赛为 2025 年全国大学生计算机系统能力大赛-编译系统设计赛-编译系统挑战赛-代码体积赛道，该赛道内容为在 RISC-V32 位架构上，修改 llvm 工具链对比赛提供的汇编文件和目标文件进行优化，在保障功能正确的前提下，尽量减小代码体积。初赛提供了 4 个公开的测试用例与源码，决赛新增 4 个保密的测试用例。

参赛队伍需要将修改后的工具链仓库地址提交到测评系统中，由测评系统自动构建与评判，给出其在各个用例上获得的分数，并公开在比赛排行榜中。

比赛官网：

https://compiler.educg.net/#/index?TYPE=2025COM_C

比赛明确规定了以下违规条例：

三、如果优化策略是基于对特定测试用例的特征（如输入数据或函数名、函数参数个数及其输入值等）的判断，那么这种优化将被判定为违规行为，具体包括：

（1）通过识别函数名或特定字符串进行特定优化：编译器匹配函数名或特定字符串来推测测试用例类型，并进行非通用的特定优化。

（2）对特定测试用例的计算结果进行硬编码：编译器识别特定测试用例中的某个函数或变量特征，直接返回结果而不进行真实计算。

（3）识别特定测试用例的输入模式并进行特定优化：编译器通过判断输入数据的特征（如大小、特定字符串）来激活特定优化路径。

（4）**利用代码未定义行为：**在编译优化中利用代码中未定义行为来获得性能提升，在其他输入条件下可能导致错误或不一致的结果。

（5）其他通过探测特定输入模式或条件、依赖于特定输入或环境、不能保证通用情况下正确性的投机性或猜测性优化。

二、违规行为证明材料

1. 识别测试用例文件名与符号名删除

aisystem 队伍通过字符串匹配，识别文件名与函数名，进行了大量非通用的优化。
其中影响最严重，完全不具有通用性的是
在 tiny/bolt/lib/Passes/RISCV/RISCVElimUnusedFuncs.cpp 文件中，通过**人工标记大量函数名**构建列表，对大量函数进行了删除，其中不乏会破坏正常功能的删除，以及对决赛未公开测试用例与符号名识别与删除等严重违规内容，**符合违规行为认定说明中的第一条**：

> **（1）通过识别函数名或特定字符串进行特定优化：**编译器匹配函数名或特定字符串来推测测试用例类型，并进行非通用的特定优化。

以下是违规行为源码分析：

tiny/bolt/lib/Passes/RISCV/RISCVElimUnusedFuncs.cpp
该文件的功能为：匹配输入文件的名称，对于匹配到的名称，应用预先人工标记的函数符号列表 funcNames，将列表中的函数进行 clear 处理，删除这些函数。

1.1 对文件名加密并识别用例名称：

在该文件的第 15 行至第 622 行，aisystem 队伍通过 if 识别输入的文件名称。

```
18  >  if (BC.getFilename() == "huffbench") {···  决赛用例
65  >  else if (BC.getFilename() == "sglib-combined") {···决赛用例
109 >      else if (EnF == "Q1K4ohS") {···    bm1
223 >      else if (EnF == "Q1L5ql[") {···    bm2
289 >      else if (EnF == "`EB6Cn~") {···    bm3
493 >      else if (EnF == "aB@JY<=") {···    4743
595 >      else {···
```

对于前两个名称，我们推测其为未公开的决赛用例名。

对于后四个名称，aisystem 队伍使用了加密混淆的方式，将文件名进行了加密，通过 encryptString 获得密文并匹配。

```
11  void RISCVElimUnusedFuncs::runOnFunction(BinaryFunction &BF) {
12      auto &BC = BF.getBinaryContext();
13      auto EnF = llvm_utils::encryptString(BC.getFilename());
14      std::vector<std::string> funcNames;
15      if (BC.getFilename() == "huffbench") {
16          funcNames = {
```

我们通过在其中增加输出信息的方式，追溯了密文与用例名的对应关系：

```
109         else if (EnF == "Q1K4ohS") {
110
111     llvm::errs() << "\n[DEBUG] " << __FILE__ << __LINE__ << ":" << " 该用例是Q1K4ohS" << "\n";
112
113             funcNames = {
```

encryptStrin 方法的实现在文件
jndxbase/tiny/bolt/include/bolt/Passes/Code.h 中

为类似凯撒加密的变种。

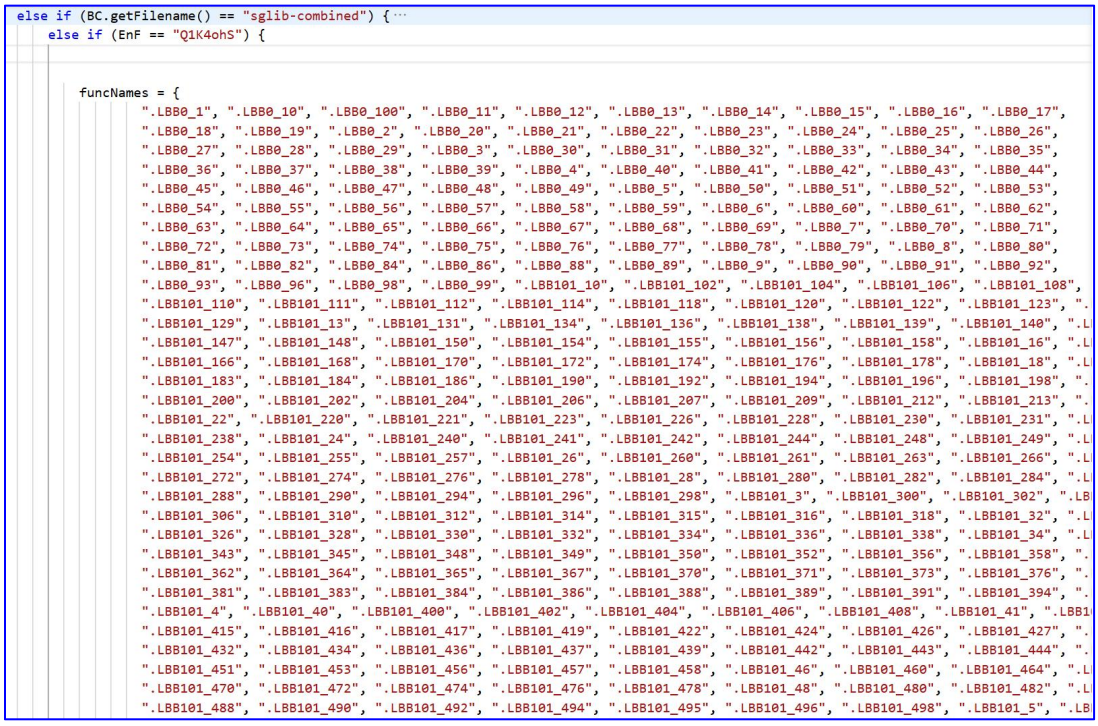
```
namespace llvm_utils {

    inline char encOne(char c, unsigned key) {
        unsigned idx = static_cast<unsigned>(c) - 32;           // 0-94
        unsigned encIdx = (idx + key) % 95;                     // 位移
        return static_cast<char>(encIdx + 32);                  // 回到 ASCII
    }

    inline char decOne(char c, unsigned key) {
        unsigned idx = static_cast<unsigned>(c) - 32;
        unsigned decIdx = (idx + 95 - (key % 95)) % 95;         // 逆位移
        return static_cast<char>(decIdx + 32);
    }

/// ★ 计算每一位的“滚动位移量”—随位置和 rolling 变化
    inline unsigned rollingKey(unsigned baseSeed,
                               unsigned rolling,
                               unsigned pos) {
        // 取一个质数 37 做增量，可自行调整
        return (baseSeed + rolling + pos * 37) & 0xFF;
    }

//————————————————————————————————
// 对外接口（std::string 版本）
//————————————————————————————————
```

aisystem 队伍使用加密来识别用例名的行为，说明他们在明知该行为违规的情况下仍然采用，并试图掩饰，十分恶劣！

1.2 人工标记的函数符号列表：

在该文件的第 15 行至第 622 行，aisystem 队伍通过人工实现 funcNames 列表匹配大量函数名。

下图是其中一个列表的部分内容：

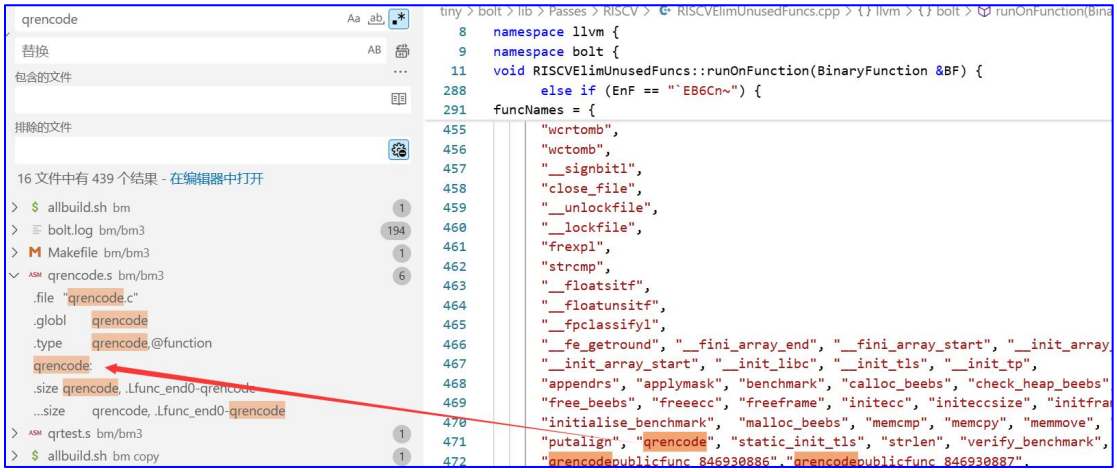


由于 aisystem 队伍**足足标注了 600 行的函数名**，便不在此一一列举，可查询其仓库 tiny/bolt/lib/Passes/RISCV/RISCVElimUnusedFuncs.cpp 文件第 15 行至第 622 行进行确认。

同时，可以确定这些符号来自测试用例中，以下是其中两个的示例：
在 EnF == "Q1K4ohS"列表中出现的符号.LBB101_255 可在 bm1.s 中找到。

```
.LBB101_255:
    lw  a1, 4(s1)
    lw  a0, 0(s1)
    mv  a2, s7
    lw  a3, 56(sp)
    call    transparent_crc
    lw  a0, 52(sp)
    bnez    a0, .LBB101_257
    mv  a0, s2
    mv  a1, s3
    mv  a2, s0
    call    printf
```

在 EnF == "`EB6Cn~"列表中出现的函数 qrencode 可以在 bm3 的源文件中找到。

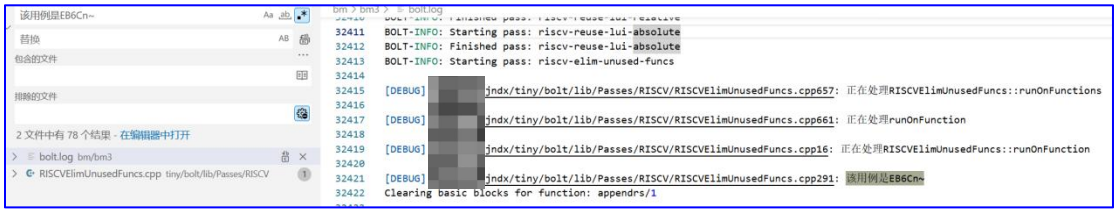


bm > bm3 > qrencode.s

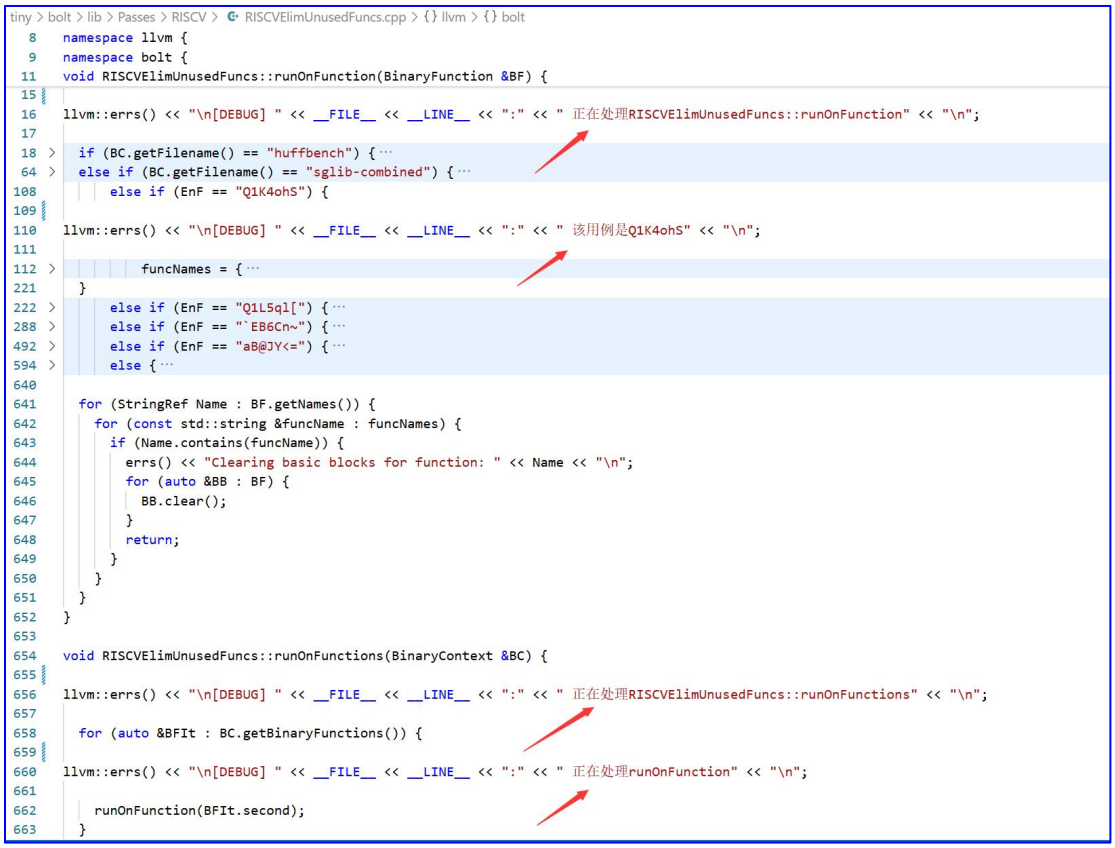
```
    .text
    .attribute  4, 16
    .attribute  5, "rv32i2p1_m2p0_f2p2_d2p2_c2p0_zicsr2p0"
    .file   "qrencode.c"
    .globl  qrencode
    .p2align    1
    .type   qrencode,@function
qrencode:
    addi    sp, sp, -64
    sw  ra, 60(sp)
    sw  s0, 56(sp)
    sw  s1, 52(sp)
    sw  s2, 48(sp)
    sw  s3, 44(sp)
    sw  s4, 40(sp)
    sw  s5, 36(sp)
    sw  s6, 32(sp)
```

1.3 对匹配到的用例中匹配到的基本块进行删除：

```
624   for (StringRef Name : BF.getNames()) {
625     for (const std::string &funcName : funcNames) {
626       if (Name.contains(funcName)) {
627         errs() << "Clearing basic blocks for function: " << Name << "\n";
628         for (auto &BB : BF) {
629           BB.clear();
630         }
631         return;
632       }
633     }
634   }
635 }
636
```



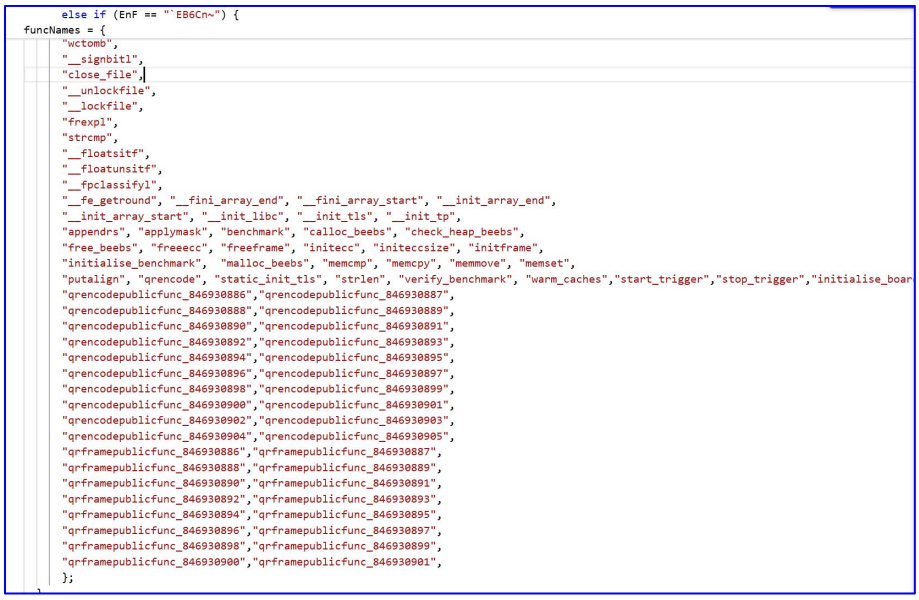
该日志信息是通过增加输出信息，运行编译用例后得到的，证明这段违规函数确实应用在了项目中。



2. 删除程序正常功能的优化，用无功能的文件伪造用例结果

aisystem 队伍通过识别函数名的方式对测试用例 bm3 进行了破坏性的删除，使程序失去了原本应有的功能。

如上节违规行为中所述，在 RISCVElimUnusedFuncs.cpp 函数中，aisystem 队伍对于 funcNames 列表中的函数名对 bm3 进行了删除处理。



```
    else if (EnF == "`EB6Cn~") {
funcNames = {
    "wctomb",
    "__signbitl",
    "close_file",
    "__unlockfile",
    "__lockfile",
    "frexpl",
    "strcmp",
    "__floatsitf",
    "__floatunsitf",
    "__fpclassifyl",
    "__fe_getround", "__fini_array_end", "__fini_array_start", "__init_array_end",
    "__init_array_start", "__init_libc", "__init_tls", "__init_tp",
    "appendrs", "applymask", "benchmark", "calloc_beebs", "check_heap_beebs",
    "free_beebs", "freeecc", "freeframe", "initecc", "initeccsize", "initframe",
    "initialise_benchmark",  "malloc_beebs", "memcmp", "memcpy", "memmove", "memset",
    "putalign", "qrencode", "static_init_tls", "strlen", "verify_benchmark", "warm_caches","start_trigger","stop_trigger","initialise_boar
    "qrencodepublicfunc_846930886","qrencodepublicfunc_846930887",
    "qrencodepublicfunc_846930888","qrencodepublicfunc_846930889",
    "qrencodepublicfunc_846930890","qrencodepublicfunc_846930891",
    "qrencodepublicfunc_846930892","qrencodepublicfunc_846930893",
    "qrencodepublicfunc_846930894","qrencodepublicfunc_846930895",
    "qrencodepublicfunc_846930896","qrencodepublicfunc_846930897",
    "qrencodepublicfunc_846930898","qrencodepublicfunc_846930899",
    "qrencodepublicfunc_846930900","qrencodepublicfunc_846930901",
    "qrencodepublicfunc_846930902","qrencodepublicfunc_846930903",
    "qrencodepublicfunc_846930904","qrencodepublicfunc_846930905",
    "qrframepublicfunc_846930886","qrframepublicfunc_846930887",
    "qrframepublicfunc_846930888","qrframepublicfunc_846930889",
    "qrframepublicfunc_846930890","qrframepublicfunc_846930891",
    "qrframepublicfunc_846930892","qrframepublicfunc_846930893",
    "qrframepublicfunc_846930894","qrframepublicfunc_846930895",
    "qrframepublicfunc_846930896","qrframepublicfunc_846930897",
    "qrframepublicfunc_846930898","qrframepublicfunc_846930899",
    "qrframepublicfunc_846930900","qrframepublicfunc_846930901",
    };
}
```

我们通过关闭 strip 保留符号表和关闭代码压缩，获取了真实的反汇编结果如下：

qrduino: file format elf32-littleriscv

Disassembly of section .local.text.init_heap_beebs:

000116dc <init_heap_beebs>:

```
    116dc: 37 26 01 00          lui    a2, 18
    116e0: a3 28 a6 7e          sw     a0, 2033(a2)
    116e4: 2e 95                add    a0, a0, a1
    116e6: b7 25 01 00          lui    a1, 18
    116ea: a3 aa a5 7e          sw     a0, 2037(a1)
    116ee: 37 25 01 00          lui    a0, 18
    116f2: a3 2c 05 7e    sw    zero, 2041(a0)
    116f6: 82 80                ret
```

Disassembly of section .local.text.main:

000116f8 <main>:

```
    116f8: 41 11                addi sp, sp, -16
    116fa: 06 c6                sw     ra, 12(sp)
    116fc: ef 00 c0 02    jal    0x11728 <_start>
    11700: ef f0 df fd    jal    0x116dc <init_heap_beebs>
    11704: 05 45                li     a0, 1
    11706: ef f0 7f fd    jal    0x116dc <init_heap_beebs>
    1170a: ef 00 e0 01          jal    0x11728 <_start>
    1170e: ef f0 ff fc    jal    0x116dc <init_heap_beebs>
    11712: 2a c4                sw     a0, 8(sp)
    11714: ef 00 40 01          jal    0x11728 <_start>
```

```
    11718: 22 45                lw    a0, 8(sp)
    1171a: ef f0 3f fc      jal   0x116dc <init_heap_beebs>
    1171e: 13 35 15 00          seqz a0, a0
    11722: b2 40                lw    ra, 12(sp)
    11724: 41 01                addi sp, sp, 16
    11726: 82 80                ret

Disassembly of section .local.text._start:

00011728 <_start>:
    11728: 97 41 00 00          auipc       gp, 4
    1172c: 93 81 01 26          addi gp, gp, 608
    11730: 0a 85                mv    a0, sp
    11732: 97 d5 fe ff      auipc       a1, 1048557
    11736: 93 85 05 00          mv    a1, a1
    1173a: 13 71 01 ff      andi sp, sp, -16
    1173e: ef 00 40 00          jal   0x11742 <_start_c>

Disassembly of section .local.text._start_c:

00011742 <_start_c>:
    11742: 0c 41                lw    a1, 0(a0)
    11744: 13 06 45 00          addi a2, a0, 4
    11748: 37 15 01 00          lui   a0, 17
    1174c: 13 05 85 6f      addi a0, a0, 1784
    11750: b7 16 01 00          lui   a3, 17
    11754: 93 86 c6 6d          addi a3, a3, 1756
    11758: 37 17 01 00          lui   a4, 17
    1175c: 13 07 c7 6d          addi a4, a4, 1756
    11760: 81 47                li    a5, 0
    11762: 6f 00 40 03          j     0x11796 <libc_start_main_stage2>

Disassembly of section .local.text.libc_start_init/1:

00011766 <libc_start_init>:
    11766: 41 11                addi sp, sp, -16
    11768: 06 c6                sw    ra, 12(sp)
    1176a: 22 c4                sw    s0, 8(sp)
    1176c: 26 c2                sw    s1, 4(sp)
    1176e: ef f0 ff f6      jal   0x116dc <init_heap_beebs>
    11772: 37 15 01 00          lui   a0, 17
    11776: 13 04 c5 6d          addi s0, a0, 1756
    1177a: 93 04 04 00          mv    s1, s0
    1177e: 63 f7 84 00          bgeu        s1, s0, 0x1178c <libc_start_init+0x26>
```

```
    11782: 88 40            lw    a0, 0(s1)
    11784: 02 95            jalr  a0
    11786: 91 04            addi s1, s1, 4
    11788: e3 ed 84 fe      bltu  s1, s0, 0x11782 <libc_start_init+0x1c>
    1178c: b2 40            lw    ra, 12(sp)
    1178e: 22 44            lw    s0, 8(sp)
    11790: 92 44            lw    s1, 4(sp)
    11792: 41 01            addi sp, sp, 16
    11794: 82 80            ret

Disassembly of section .local.text.libc_start_main_stage2/1:

00011796 <libc_start_main_stage2>:
    11796: 01 11            addi sp, sp, -32
    11798: 06 ce            sw    ra, 28(sp)
    1179a: 22 cc            sw    s0, 24(sp)
    1179c: 26 ca            sw    s1, 20(sp)
    1179e: 4a c8            sw    s2, 16(sp)
    117a0: 4e c6            sw    s3, 12(sp)
    117a2: 32 84            mv    s0, a2
    117a4: ae 84            mv    s1, a1
    117a6: 2a 89            mv    s2, a0
    117a8: 13 95 25 00      slli  a0, a1, 2
    117ac: 32 95            add   a0, a0, a2
    117ae: 93 09 45 00      addi s3, a0, 4
    117b2: ef f0 5f fb      jal   0x11766 <libc_start_init>
    117b6: 26 85            mv    a0, s1
    117b8: a2 85            mv    a1, s0
    117ba: 4e 86            mv    a2, s3
    117bc: 02 99            jalr  s2
    117be: ef 00 40 00      jal   0x117c2 <_Exit>

Disassembly of section .local.text._Exit:

000117c2 <_Exit>:
    117c2: aa 85            mv    a1, a0
    117c4: 93 08 e0 05      li    a7, 94
    117c8: 73 00 00 00      ecall
    117cc: 93 08 d0 05      li    a7, 93
    117d0: 2e 85            mv    a0, a1
    117d2: 73 00 00 00      ecall
    117d6: ed bf            j     0x117d0 <_Exit+0xe>
```

该反汇编结果显示，aisystem 队伍的“优化”只保留了入口和出口，将原程序主要功能函数

qrencode 等都进行了删除，与该程序原本的二维码校验功能完全不符。

aisystem 队伍的方法利用了 bm3 测试用例只进行计算验证而没有输出的特性，通过识别文件名和函数名对程序进行了破坏性的删除，不进行真实计算，**符合违规行为认定说明中的第一条与第二条**。

> **（1）通过识别函数名或特定字符串进行特定优化：**编译器匹配函数名或特定字符串来推测测试用例类型，并进行非通用的特定优化。
>
> **（2）对特定测试用例的计算结果进行硬编码：**编译器识别特定测试用例中的某个函数或变量特征，直接返回结果而不进行真实计算。

以下是复现该反汇编结果的指引：

由于 aisystem 队伍使用的 strip 和文件压缩可能会对符号表和反汇编结果造成影响，需要将其关闭。
在 tiny/lld/ELF/Driver.cpp 中的 static void runBoltAndPack
函数包含了这些的调用。

将 2836 行注释掉，可关闭 bolt 对符号表的删除：

```
2828      cmd = q(boltBin) + ' ' + q(objPath) +
2829        " --use-old-text"
2830        " --align-text=0x2"
2831        " -v 3"
2832        " --align-functions=0x2"
2833        " --icf " + q(objPath) +
2834        " -o " + q(outFile) +
2835        " --ignore-build-id"
2836        // " --remove-symtab"
2837        " --mini-rodata"
2838        " --use-gnu-stack > bolt.log 2>&1";
2839      if (std::system(cmd.c_str()) != 0) return;
2840      if (!waitForFile(outFile)) return;
```

将 2850 行至 2857 行注释掉，可关闭 strip 功能，保留符号表：

```
2850    // // 5. strip final
2851    // cmd = q(objcopyBin) +
2852    //       " --strip-all --strip-sections"
2853    //       " --remove-section=.gnu.stack"
2854    //       " --remove-section=.riscv.attributes"
2855    //       " --remove-section=.note.gnu.build-id "
2856    //       + q(finalFile) + " > /dev/null 2>&1";
2857    // std::system(cmd.c_str());
```

将 2868 行至 2875 行注释掉，可关闭压缩功能：

```
2868    // // 7. 判断大小并尝试 pack payload
2869    // if (::stat(objPath.c_str(), &st) != 0) {
2870    //   std::fprintf(stderr, "Failed to stat %s: %s\n", objPath.c_str(), strerror(errno));
2871    //   return;
2872    // }
2873    // if (objPath == "bm1.out" || objPath == "bm2.out" || objPath == "qrduino" || objPath ==
2874    //     if (st.st_size > 50000 && !buildPayload(objPath))
2875    //       std::fprintf(stderr, "payload build failed\n");
```

关闭这三个功能后，使用比赛的 build 指令对项目进行构建，使用比赛的编译指令对 bm3 进行编译。
使用以下指令生成编译后文件的反汇编结果：
llvm-objdump -d qrduino > qrduino_d.s

该创新使 aisystem 队伍获得了遥遥领先的体积分数和性能分数，官方没有产生任何质疑且授予其特等奖表彰。

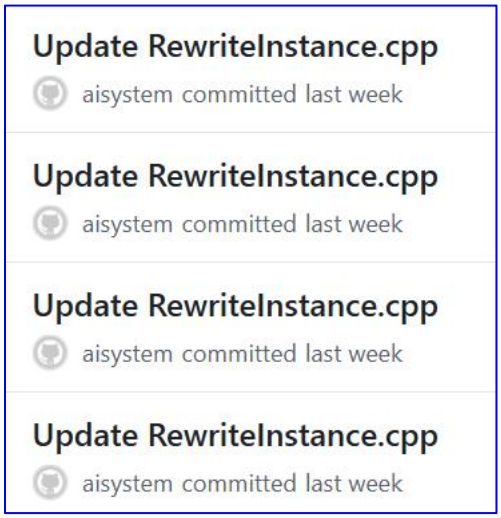
BM3_codesize_score	BM3_func_score	BM3_perf_score
441.902	100	5
199.277	100	-10
143.511	100	-10
165.782	100	-10
131.413	100	-10

3. 通过硬编码调整偏移量，进行针对测评系统的优化，在其他环境会报错。

在文件 tiny/bolt/lib/Rewrite/RewriteInstance.cpp 第 3508 行
aisystem 队伍提出了极其抽象的手工调参方法。

```
////////////////////////////////////////////////////////////////////////////////
// Assign addresses to new sections.
////////////////////////////////////////////////////////////////////////////////

// Get output object as ObjectFile.
std::unique_ptr<MemoryBuffer> ObjectMemBuffer =
    MemoryBuffer::getMemBuffer(ObjectBuffer, "in-memory object file", false);

auto EFMM = std::make_unique<ExecutableFileMemoryManager>(*BC);
EFMM->setNewSecPrefix(getNewSecPrefix());
EFMM->setOrgSecPrefix(getOrgSecPrefix());
ErrorOr<BinarySection &> DataSection = BC->getUniqueSectionByName(".data");
if (!DataSection)
  DataSection = BC->getUniqueSectionByName(".sdata");  // NOTE bm3没有.data会段错!
Linker = std::make_unique<JITLinkLinker>(*BC, std::move(EFMM));
// 因为 BC->NewTextSectionEnd尚不确定，所以应当执行pass时记录缩小大小!
  errs() << "SIZECHANGE: " << BC->sizeChange << "\n";
  std::string fname = BC->getFilename().str();

  if (fname.find("bm6") != std::string::npos)    // ⚠ ⚠ ⚠ ⚠
      BC->sizeChange = 0x5000;    //0x4800
  else if (fname.find("bm1") != std::string::npos)
      BC->sizeChange = 0x5352;
  else if (fname.find("bm2") != std::string::npos)    // 无需用压缩壳的不要修改
      BC->sizeChange = 0x4D76;
  else if (fname.find("qrduino") != std::string::npos)   // ⚠ 注意llvm-objcopy可能导致BUG
      BC->sizeChange = 0x29c7;  // 28ab
  else if (fname.find("rnd") != std::string::npos) // 注意 4826 也是 rnd.out!
      BC->sizeChange = 0x4b00; // 原5188(最新的是5196) 最好0x5198
  else if (fname.find("huffbench") != std::string::npos)
      BC->sizeChange = 0x7b0; // 240  2a0
  else if (fname.find("sglib-combined") != std::string::npos)
      BC->sizeChange = 0x6e0; // 240  280
  if (opts::ManualOffset != 0x0) {
      llvm::outs() << "ManualOffsetSetAt:" << opts::ManualOffset << "\n";
      BC->sizeChange = opts::ManualOffset;
  }
```

该队伍仓库的最新几次提交显示，他们在反复修改 sizeChange 的数字。



```
3527        -           BC->sizeChange = 0x4d00;    //0x4800
      3527  +           BC->sizeChange = 0x5000;    //0x4800
3528  3528          else if (fname.find("bm1") != std::string::npos)
3529  3529              BC->sizeChange = 0x5352;
3530  3530          else if (fname.find("bm2") != std::string::npos)    // 无需用压缩壳的不要修改
3531  3531              BC->sizeChange = 0x4D76;
3532  3532          else if (fname.find("qrduino") != std::string::npos)   // ⚠️ 注意llvm-objcopy可能导致BUG
3533  3533              BC->sizeChange = 0x29c7;  // 28ab
3534  3534          else if (fname.find("rnd") != std::string::npos) // 注意 4826 也是 rnd.out!
3535        -           BC->sizeChange = 0x4a00; // 原5188(最新的是5196) 最好0x5198
      3535  +           BC->sizeChange = 0x4b00; // 原5188(最新的是5196) 最好0x5198
3536  3536          else if (fname.find("huffbench") != std::string::npos)
3537  3537              BC->sizeChange = 0x7b0; // 240  2a0
3538  3538          else if (fname.find("sglib-combined") != std::string::npos)
3539        -           BC->sizeChange = 0x6c0; // 240  280
      3539  +           BC->sizeChange = 0x6e0; // 240  280
```

测试发现，随着设置的数字不断减小，生成的文件体积也不断变小，在小到一定程度后会报错。同时，报错阈值在不同环境下表现不同，他们最终的设置在测评系统中能够正常运行，但在本地 qemu8 中报错 3 个，在 qemu6 中报错 1 个。

```
./run_rnd.sh: line 3:  9857 Segmentation fault      (core dumped) qemu-riscv32 -B 0x10000000 ./rnd.out > rnd.txt
1d0
< checksum = 966B3545
fail
./run_bm1.sh: line 3:  9867 Segmentation fault      (core dumped) qemu-riscv32 -B 0x10000000 ./bm1.out > output.txt
1d0
< checksum = 51CB0F9A
fail
pass
./run_bm3.sh: line 9:  9889 Segmentation fault      (core dumped) qemu-riscv32 -B 0x10000000 ./qrduino
fail
```

推测是用于切断 bolt 优化后多余的冗余空间，但是这个冗余在不同地方表现不同，aisystem 队伍直接硬编码了 sizeChange 对内存布局进行切断，在测评系统里刚好卡在边缘就正常，在其他环境切到代码节则产生了报错。

4. 除此之外，还有以下一些问题：

4.1

在 clang 中，aisystem 队伍也进行了类似的文件名匹配和人工符号列表匹配删除的操作。tiny/clang/tools/driver/RISCVPreMCOptimizer.cpp 第 24 行定义列表：

```
static const int sec_mys_i[] = {
    104,   311,   312,   313,   1467,  2286,  2460,  2461,  2719,  2747,  3049,
    3728,  3729,  4143,  4566,  5072,  5073, /* 5116,  */5176,
    5403,  5404,  5405,  6307,  6308,  6309,  7721,  7722,  7725,  7881,  7882,
    7883,  8124,  8125,  8875,  9284,  9576,  10333, 10435, 10777, 10988, 10989,
    10990, 10991, 12225,/* 12226,*/ 12524, 12527, 12677, 12678, 13431, 13432, 13437,
    13439, 13774, 13801, 13886, 13887, 13889, 14005, 14006};

static const int sec_mys_ii[] = {199, 524, 555, 901};

static const int sec_mys_iii[] = {1895, 2136};

static const char *del_mys_i[] = {
    ".L__const.func_40.l_1066", ".L__const.func_40.l_1092",
    ".L__const.func_40.l_1123", ".L__const.func_40.l_1230",
    ".L__const.func_40.l_1281", ".L__const.func_40.l_1322",
    ".L__const.func_40.l_1323", ".L__const.func_40.l_1413",
    ".L__const.func_40.l_1581", ".L__const.func_40.l_1661",
    ".L__const.func_40.l_1665", ".L__const.func_40.l_1666",
    ".L__const.func_40.l_1691", ".L__const.func_40.l_490",
    ".L__const.func_40.l_529",  ".L__const.func_40.l_966",
    ".L__const.func_40.tmp",    ".L__const.func_57.l_207",
    ".L__const.func_40.l_1691", ".L__const.func_40.l_1665",
};

static const char *del_mys_ii[] = {
    ".L__const.func_36.l_1945", ".L__const.func_36.l_2275",
    ".L__const.func_36.l_2325", ".L__const.func_36.l_2436",
    ".L__const.func_36.l_2476", ".L__const.func_36.l_2477",
    ".L__const.func_80.l_145",  ".L__const.func_80.l_180",
    ".L__const.func_80.l_186"};
```

tiny/clang/tools/driver/RISCVPreMCOptimizer.cpp 第 440 行定义函数：

```
void RISCVPreMCOptimizer::removeWeakSymbols(const std::string &fileName,
                                            StringVec &RV32Insns) {
  // 1) 根据 fileName 选择要删除的符号数组
  const int *gArr = nullptr;
  size_t gN = 0;
  const char *const *cArr = nullptr;
  size_t cN = 0;

  // 如果 fileName 带路径或后缀，可在此处做一次规范化：
  // std::string base = llvm::sys::path::filename(fileName);
  // auto pos = base.rfind('.');
  // if (pos != std::string::npos) base.resize(pos);

  if (fileName == "bm1") {
    gArr = sec_mys_i;
    gN = sizeof(sec_mys_i) / sizeof(sec_mys_i[0]);
  } else if (fileName == "47431") {
    gArr = sec_mys_ii;
    gN = sizeof(sec_mys_ii) / sizeof(sec_mys_ii[0]);
    cArr = del_mys_i;
    cN = sizeof(del_mys_i) / sizeof(del_mys_i[0]);
  } else if (fileName == "47432") {
    gArr = sec_mys_iii;
    gN = sizeof(sec_mys_iii) / sizeof(sec_mys_iii[0]);
    cArr = del_mys_ii;
    cN = sizeof(del_mys_ii) / sizeof(del_mys_ii[0]);
  } else {
```

tiny/clang/tools/driver/cc1as_main.cpp 第 409 行调用上述函数：
该处的==0 是匹配字符串在开头位置的意思，并非关闭。

```
RISCVPreMCOptimizer RVOptimizer(Opts.InputFile, std::move(*Buffer));
  /**
   * for (const auto &line : RVOptimizer.RV32Insns) {
   * std::cout << line << std::endl;
   * }
  **/
if (Opts.InputFile.find("bm1") == 0)
  RVOptimizer.removeWeakSymbols("bm1", RVOptimizer.RV32Insns); // 删除弱符号
else if (Opts.InputFile.find("bm2_4") == 0)
  RVOptimizer.simplifyLoop(RVOptimizer.RV32Insns);
else if (Opts.InputFile.find("rnd_output0") == 0)             // ⚠️决赛关闭
  RVOptimizer.removeWeakSymbols("47431", RVOptimizer.RV32Insns);
else if (Opts.InputFile.find("rnd_output1") == 0)             // ⚠️决赛关闭
  RVOptimizer.removeWeakSymbols("47432", RVOptimizer.RV32Insns);

int Number=0;
if (Opts.InputFile.find("bm1") == 0 || Opts.InputFile.find("bm2") == 0  || Opts.InputFile.find("bm4") == 0)
  Number=1;
else if (Opts.InputFile.find("rnd") == 0 || Opts.InputFile.find("bm5") == 0 || Opts.InputFile.find("bm6") == 0)
  Number=2;   // ⚠️⚠️⚠️bm5,bm6真的不可以走Number = 1吗?
if(Number == 1) {
  // if (Opts.InputFile.find("bm1") == 0) {      // ⚠️⚠️ bm6可能也可以走这个优化!
  if (Opts.InputFile.find("bm1") == 0 || Opts.InputFile.find("bm6") == 0) {
    RVOptimizer.buildCompactSymbolMap(RVOptimizer.RV32Insns); // 构建符号映射
    RVOptimizer.applySymbolRemap(RVOptimizer.RV32Insns);      // 替换符号
  }
  RVOptimizer.removeAMSSections(RVOptimizer.RV32Insns);
  RVOptimizer.performOptimization();
  RVOptimizer.saveAndRestoreRA(RVOptimizer.RV32Insns);
  RVOptimizer.undoReplacement(RVOptimizer.RV32Insns);
  RVOptimizer.insertFunctionSections(RVOptimizer.RV32Insns);
  RVOptimizer.optimizeStackPush();
  RVOptimizer.replaceStackWithTp();
  RVOptimizer.removeUnusedfunc();
} else {
```

还有许多其他根据文件名匹配进行的设置，可在项目中搜索 bm2.out 查看，不再一一列出。根据注释中的时间来看，该队伍早在 25 年 4 月份就开始进行违规优化了。

```
if (BC.isRISCV() || BC.isRISCV32()) {
  Manager.registerPass(std::make_unique<FixRISCVCallsPass>(PrintFixRISCVCalls));
  Manager.registerPass(std::make_unique<RISCVAnalyze>(NeverPrint));
  Manager.registerPass(std::make_unique<FrameOptimizerPass>(PrintFOP));
  Manager.registerPass(std::make_unique<IdenticalCodeFolding>(PrintICF), true);
  // YANGZI20250407
  Manager.registerPass(std::make_unique<RISCVStackPushOpt>(NeverPrint));
  // YANGZI20250411
  Manager.registerPass(std::make_unique<RISCVReuseLUIAggressive>(NeverPrint));
  if (BC.getFilename() == "rnd.out" ) //      YANGZI20250416,20
    Manager.registerPass(std::make_unique<RISCVMemorySSA>(NeverPrint));        // 对bm5有BUG!无限运行
  // YANGZI20250416
  Manager.registerPass(std::make_unique<RISCVReuseLUIRelative>(NeverPrint));
  // YANGZI20250403
  Manager.registerPass(std::make_unique<RISCVReuseLUIAbsolute>(NeverPrint));
  // YANGZI20250403
  // FIXME 危险操作
  Manager.registerPass(std::make_unique<RISCVElimUnusedFuncs>(NeverPrint));
  // TODO 参考ReorderData实现删减data，因为其基于MemoryProfiling
  Manager.registerPass(std::make_unique<ReorderData>());
  // Manager.registerPass(std::make_unique<LoopInversionPass>());
  // Manager.registerPass(std::make_unique<Peepholes>(PrintPeepholes));
  // 下面为无效优化
  // Manager.registerPass(std::make_unique<SimplifyRODataLoads>(PrintSimplifyROLoads), true);
  // Manager.registerPass(std::make_unique<EliminateUnreachableBlocks>(PrintUCE), true);
  // Manager.registerPass(std::make_unique<RemoveNops>(NeverPrint));
  // Manager.registerPass(std::make_unique<InlineMemcpy>(NeverPrint), opts::StringOps);
  // Manager.registerPass(std::make_unique<Inliner>(PrintInline));
  goto goon;
```

4.2

aisystem 队伍在代码中频繁出现的 huffbench 和 sglib-combined 两个文件名，且备注标明了 4826 用例的产物名，对于这些未公开的决赛用例该队伍如何得知，还需要一个合理的解释。

```
// 注意 4826 也是 rnd.out!
```

4.3

tiny/bolt/lib/Passes/RISCV/RISCVAnalyze.cpp 中说明可能未按要求使用比赛提供的 llvm 项目版本。

```
1   //===-- RISCVAnalyze.cpp - RISC-V 专用分析 Pass 示例 ----------------===//
2   //
3   //   本示例演示如何：
4   //   1. 从 BinaryFunction 所属的 BinaryContext 中直接拿到 MCContext、
5   //      MCSubtargetInfo、MCInstrInfo、MCRegisterInfo 等对象指针；
6   //   2. 通过 getLayout().blocks() 遍历所有 BasicBlock（而非私有的 layout()）；
7   //   3. 遍历每条 MCInst，打印 opcode、操作数类型，并查询调度延迟（Latency）；
8   //   4. 演示了如何在 LLVM 18.1.2 及以后版本中正确使用 BOLT API，
9   //      并在没有调度模型时跳过延迟计算，避免断言崩溃。
10  //
11  //===--------------------------------------------------------------------===//
12  /*
```

4.4

在决赛过程中，答辩现场的专家在其他队伍答辩时提及“江南大学的优化也是通用的”进行干扰，并询问是否与江南大学有过交流。以及颁奖时有江南大学专家参与颁奖。对是否有利益相关专家在答辩现场，以及为何未采取回避制度仍抱有疑问，要求公开决赛答辩现场视频并进行审查。