Pa1 实验报告

241300015 秦浩哲

1. 计算 1+2+...+100 的程序状态机

程序逻辑：初始化`sum=0`、`i=1`，循环累加`sum += i`并递增`i`，直到`i>100`后结束。
状态机如下：

状态定义：

S0（初始状态）：初始化`sum=0`，`i=1`，完成后进入 S1。

S1（判断状态）：检查`i <= 100`？

- 是：进入 S2；
- 否：进入 S3。

S2（累加状态）：执行`sum += i`，`i += 1`，完成后返回 S1。

S3（结束状态）：程序终止。

状态转换图：

```
S0 → S1
S1 ┬→ S2（i≤100）
   └→ S3（i>100）
S2 → S1
```

2. 调试时间计算

（1）无简易调试器的总调试时间

调试次数：500 次编译 × 90% = 450 次调试。

每次调试需获取 20 个信息，每个信息耗时 30 秒：

总时间 = 450 次 × 20 个信息 × 30 秒/信息 = 270,000 秒 = 75 小时。

2）有简易调试器的节省时间

每个信息耗时 10 秒，总时间 = 450 × 20 × 10 = 90,000 秒 = 25 小时。

节省时间 = 75 小时 - 25 小时 = 50 小时。

3. ISA 手册查阅范围（RTFM）

RISC-V32

指令格式：参考 The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA，第 2 章“Base Instruction Formats”，包括 R（寄存器）、I（立即数）、S（存储）、B（分支）、U（高位立即数）、J（跳转）6 种格式。

LUI 指令：参考第 3 章“Load and Store Instructions”，LUI（Load Upper Immediate）将 20 位立即数加载到寄存器的高 20 位，低 12 位清 0（`rd = imm[31:12] << 12`）。

mstatus 寄存器：参考 Volume II: Privileged Architecture，第 3 章“Machine-Level CSRs”，包含中断使能（如 UIE、SIE）、特权级状态（如 MPP）等字段。

4. 代码行数统计（shell 命令）

```
Switched to branch 'pa1'
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ make count
find ./ -name "*.c" -o -name "*.h" | xargs cat | wc -l
266808
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ git checkout pa0
M       nemu/Makefile
Switched to branch 'pa0'
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ make count
find ./ -name "*.c" -o -name "*.h" | xargs cat | wc -l
267350
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ git branch
  master
* pa0
  pa1
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ make count-non-empty
find ./ -name "*.c" -o -name "*.h" | xargs cat | grep -v '^[[:space:]]*$' | wc -l
grep: (standard input): binary file matches
230749
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ git checkout pa1
warning: unable to rmdir 'am-kernels': Directory not empty
M       nemu/Makefile
Switched to branch 'pa1'
hook@LAPTOP-KCNIV79K:~/ics2025/nemu$ make count-non-empty
find ./ -name "*.c" -o -name "*.h" | xargs cat | grep -v '^[[:space:]]*$' | wc -l
230381
```

```
```

5. gcc 编译选项（-Wall 和-Werror）

Wall：开启所有常见警告（如未使用的变量、类型不匹配、隐式声明等），帮助开发者发现潜在错误。

Werror：将所有警告视为错误，强制编译终止，确保开发者必须修正所有警告，避免因忽略警告导致的 bug，提高代码健壮性。　使用原因：通过严格检查编译警告，提前暴露代码中的潜在问题，减少运行时错误，提升代码质量。

手册（尤其是 ISA 手册、工具文档）往往篇幅庞大（动辄数百页），通读不仅低效，还会因细节过多导致重点模糊。第一步应先快速浏览目录，标记核心模块（如 x86 的“寄存器”“指令格式”“内存寻址”，RISC-V 的“指令集”“特权级”等），建立“知识地图”—— 即知道“某类问题大概属于哪个章节”。

例如在 PA1 实现寄存器解析时，需要知道 RISC-V 的寄存器命名规则（如 t0 对应 x5），通过目录定位到“General-Purpose Registers”章节，直接找到寄存器编号与名称的对应表，比通读全书高效 10 倍。反之，若一开始就逐页阅读，很可能错过关键章节，浪费时间

阅读手册的核心是“为我所用”—— 用目录建立框架，用问题驱动搜索，用试错调整策略，用实践强化记忆。PA1 的经历反复证明：与其因“手册太厚”而畏难，不如带着具体问题“精准出击”。这种能力不仅适用于 ISA 手册，也适用于任何技术文档（如 Makefile 手册、Git 手册），是工程实践中最核心的“生存技能”之一。

在实现表达式求值的过程中，我遇到了多方面的困难：首先是表达式生成器相关问题，包括临时文件/tmp/.code.c 未正确生成导致编译失败，函数声明顺序错误引发隐式声明冲突；其次是生成的表达式在编译时频繁出现整数溢出警告，即便缩小随机数范围仍未完全解决；此外，在表达式解析逻辑中，还存在运算优先级处理不当、负号与减号区分不清、括号匹配问题，以及枚举类型定义时的语法错误（如缺少逗号）等，这些都导致了编译错误或求值逻辑异常。

在实现 NEMU 监视点功能时，我主要遇到了四类困难：一是表达式求值阶段的解析失败问题，如设置`$t0`、`0x1000`等表达式时提示“Invalid expression”，原因是`eval`函数中常量、寄存器解析成功后未显式设置`*success = true`，且`isa_reg_str2val`函数未实现寄存器名到值的映射，解决方式是在`eval`的整数、寄存器、运算符计算等成功分支中补设`*success = true`，并完善`isa_reg_str2val`使其遍历`regs`数组匹配寄存器名并返回对应`cpu.gpr`值；二是`Makefile`中代码行数统计命令的语法错误，如`count-non-empty`目标下`grep`命令引号未闭合、命令前用空格而非 Tab 缩进导致报错，通过补全`grep`正则表达式的闭合单引号（`'^[[:space:]]*$'`）并改用 Tab 缩进解决；三是 Git 分

支切换时`am-kernels`目录非空的警告，因目录含未跟踪的临时文件导致 Git 无法删除，通过删除或备份该目录下无用文件（如编译产物），使分支切换时 Git 能正常处理目录状态；四是监视点链表管理的潜在逻辑问题，如`new_wp`分配节点、`free_wp`释放节点时可能出现的链表指针指向错误，通过确保从空闲链表取节点后更新`free_`指针、将节点加入使用中链表时正确处理`head`指针，以及删除时遍历链表找到对应节点并调整前后指针，保证监视点创建与删除功能正常。