PA1实验报告

241051011 何翌闻

实验心得：

已完成所有实验内容

在这次实验中，比较困难的部分是：

1. 负数的实现

   对于`negative`符号来说，识别并不困难，但是我必须考虑到一个数可以是诸如`-------1`这种嵌套多个负号的类型，我本来想在优先级识别的地方把负号和其他符号一起运算，但是后面想出了一个惊为天人（哪有这样夸自己的）的方法：

   ```
   int op=op_pos(p,q);

   if(op==-1){
     int count=0;
     int init=p;
     while(tokens[init].type==TK_NEG){
       count++;
       init++;
     }

     if(count%2==0) return eval(init,q,success);
     else return -eval(init,q,success);
   ```

   `op`在`op_pos()`函数中被初始化为`-1`，`return -1`的含义是没有找到除负号外的任何运算符，由于只要找到运算符就不会`return -1`，所以最后一定只剩负号！

2. 自动测试代码

   在这一部分我大量STFW了，因为在CPL中并没有学过如何读入文件

总体来说，PA1并没有我想象中那么困难

必做题：

1.程序是个状态机

```
(0, x, x) → (1, 0, x) → (2, 0, 1) → (3, 0, 1)
→ (4, 0, 1) → (5, 1, 1) → (6, 1, 2) → (2, 1, 2)
→ (3, 1, 2) → (4, 1, 2) → (5, 3, 2) → (6, 3, 3) → (2, 3, 3)
...
→ (2, 4851, 99) → (3, 4851, 99) → (4, 4851, 99) → (5, 4950, 99) → (6, 4950, 100) → (2, 4950,
100)
→ (3, 4950, 100) → (4, 4950, 100) → (5, 5050, 100) → (6, 5050, 101) → (2, 5050, 101)
→ (3, 5050, 101) → (7, 5050, 101)
```

2.调试

- 大概75小时
    1. 90%的次数是用于调试，即调试次数为：500 × 90% = 450次。
    2. 排除一个bug所需的时间为：20 × 30秒 = 600秒 = 10分钟。
    3. 总调试时间为：450 × 10分钟 = 4500分钟。
- 节省50小时
    1. 每次调试中，获取并分析一个信息只需要10秒。
    2. 排除一个bug仍然需要获取并分析20个信息。
    3. 因此，排除一个bug所需的时间为：20 × 10秒 = 200秒。

3.科学查阅手册

1. **RISC-V 32（RV32I）有哪几种指令格式?**

   Volume I

   2.2 Base Instruction Formats（Page 33）

   2.3 Immediate Encoding Variants（Page 34）

2. **LUI 指令的行为是什么?**

   Volume I

   2.4 Integer Computational Instructions（Page 36–37）

3. **mstatus寄存器的结构是怎么样的?**
   Volume II

   Section 3.1.6: Machine Status Registers (mstatus and mstatush)（ Page 34–35 ）

4.shell

利用terminal命令：

```
find nemu/ -name "*.c" -o -name "*.h" | xargs wc -l
```

```
   725 nemu/tools/capstone/repo/arch/X86/X86DisassemblerDecoder.h
    96 nemu/tools/capstone/repo/arch/X86/X86Mapping.h
    94 nemu/tools/capstone/repo/arch/X86/X86Module.c
   483 nemu/tools/capstone/repo/arch/X86/X86DisassemblerDecoderCommon.h
  2266 nemu/tools/capstone/repo/arch/X86/X86Mapping.c
    28 nemu/tools/capstone/repo/arch/X86/X86Disassembler.h
  1061 nemu/tools/capstone/repo/arch/X86/X86IntelInstPrinter.c
  2358 nemu/tools/capstone/repo/arch/X86/X86DisassemblerDecoder.c
    28 nemu/tools/capstone/repo/arch/AArch64/AArch64InstPrinter.h
    43 nemu/tools/capstone/repo/arch/AArch64/AArch64Mapping.h
   585 nemu/tools/capstone/repo/arch/AArch64/AArch64BaseInfo.h
    12 nemu/tools/capstone/repo/arch/AArch64/AArch64Module.h
  2280 nemu/tools/capstone/repo/arch/AArch64/AArch64Disassembler.c
   945 nemu/tools/capstone/repo/arch/AArch64/AArch64AddressingModes.h
  3029 nemu/tools/capstone/repo/arch/AArch64/AArch64InstPrinter.c
    44 nemu/tools/capstone/repo/arch/AArch64/AArch64Module.c
    77 nemu/tools/capstone/repo/arch/AArch64/AArch64BaseInfo.c
    18 nemu/tools/capstone/repo/arch/AArch64/AArch64Disassembler.h
   883 nemu/tools/capstone/repo/arch/AArch64/AArch64Mapping.c
   254 nemu/tools/capstone/repo/Mapping.c
   151 nemu/tools/capstone/repo/MCRegisterInfo.c
   886 nemu/tools/capstone/repo/cs_simple_types.h
267271 总计
heyween@heyween-virtual-machine:~/Desktop/ics2025$
```

总计267271行代码

接着，利用

```
git checkout pa0
find nemu/ -name "*.c" -o -name "*.h" | xargs wc -l | tail -1

git checkout pa1
find nemu/ -name "*.c" -o -name "*.h" | xargs wc -l | tail -1
```

得到：

```
切换到分支 'pa0'
 266807 总计
切换到分支 'pa1'
 267271 总计
heyween@heyween-virtual-machine:~/Desktop/ics2025$
```

再作差，就能得到pa1比pa0多了464行代码

接着，输入以下命令，可以得到除了空行外有多少代码

```
find nemu/ -name "*.c" -o -name "*.h" | xargs grep -v "^$" | wc -l
```

```
heyween@heyween-virtual-machine:~/Desktop/ics2025$ find nemu/ -name "*.c" -o -name "*.h" | xargs gre
p -v "^$" | wc -l
230952
```

最后，写入`makefile`内，就能利用`make count`和`make count_no_empty`进行查询

```
STUID = 241051011
STUNAME = 何翌闻

DO NOT modify the following code!!!

GITFLAGS = -q --author='tracer-ics2025 <tracer@njuics.org>' --no-verify --allow-empty

prototype: git_commit(msg)

define git_commit
    -@git add $(NEMU_HOME)/.. -A --ignore-errors
    -@while (test -e .git/index.lock); do sleep 0.1; done
    -@(echo "> $(1)" && echo $(STUID) $(STUNAME) && uname -a && uptime) | git commit -F -
$(GITFLAGS)
    -@sync
endef

_default:
    @echo "Please run 'make' under subprojects."

submit:
    git gc
    STUID=$(STUID) STUNAME=$(STUNAME) bash -c "$$(curl -s
http://why.ink:8080/static/submit.sh)"

.PHONY: default submit

count:
    find nemu/ -name "*.c" -o -name "*.h" | xargs wc -l

count_no_empty:
    find nemu/ -name "*.c" -o -name "*.h" | xargs grep -v "^$" | wc -l
```

5. `-Wall` 和 `-Werror` 的作用：

1. `-Wall`（启用所有警告）

- **作用**：启用gcc的大部分常用警告信息。`-Wall` 并不是真的"所有警告"，而是"大多数有用的警告"。
- **包含的警告**：包括未使用的变量、未初始化的变量、函数声明不匹配、可疑的类型转换等常见问题。
- **好处**：在编译阶段发现潜在的代码问题，提高代码质量。

2. `-Werror`（将警告视为错误）

- **作用**：将所有警告当作错误来处理。如果编译过程中产生任何警告，编译会失败。
- **好处**：强制开发者必须解决所有警告，确保代码的严谨性。