PA1 实验报告

14307130078 张博洋

一、实验进度

完成了 PA1 的所有内容。

二、必答题

1. 查阅 i386 手册

（1）EFLAGS 寄存器中的 CF 位是什么意思

首先要搞清楚 EFLAGS 是什么，因此全文检索"EFLAGS"，经过一番筛选发现第 2.3.4 节（第 33 页）详细讲解了什么是 EFLAGS 寄存器。

> **2.3.4 Flags Register**
> The flags register is a 32-bit register named EFLAGS. Figure 2-8 defines the bits within this register. The flags control certain operations and indicate the status of the 80386.

搞清楚了什么是 EFLAGS 寄存器，接下来的问题就是 CF 位是什么。继续阅读该节内容，发现手册中指出，详细的定义在附录 C 中。

> **2.3.4.1 Status Flags**
> The status flags of the EFLAGS register allow the results of one instruction to influence later instructions. The arithmetic instructions use OF, SF, ZF, AF, PF, and CF. The SCAS (Scan String), CMPS (Compare String), and LOOP instructions use ZF to signal that their operations are complete. There are instructions to set, clear, and complement CF before execution of an arithmetic instruction. **Refer to Appendix C for definition of each status flag.**

阅读附录 C 后得到 CF 位的定义。

> Carry Flag -- **Set on high-order bit carry or borrow; cleared otherwise.**

（2）ModR/M 字节是什么

全文检索"ModR/M"，经过筛选，发现在第 17.2.1 节（第 241 节）详细讲解了 ModR/M 字节，了解到 ModR/M 字节是指令格式的一部分。

> **The ModR/M byte contains three fields of information:**
> **1.** The mod field, which occupies the two most significant bits of the byte, combines with the r/m field to form 32 possible values: eight registers and 24 indexing modes.
> **2.** The reg field, which occupies the next three bits following the mod field, specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the first (opcode) byte of the instruction.
> **3.** The r/m field, which occupies the three least significant bits of the byte, can specify a register as the location of an operand, or can form part of the addressing-mode encoding in combination with the field as described above.

（3）mov 指令的具体格式是怎么样的

查看目录发现 17.2.2.11 节详细讲解了各种指令的格式。在 347 页，有 MOV 指令具体的解释。

MOV -- Move Data			
Opcode	**Instruction**	**Clocks**	**Description**
88 /r	MOV r/m8,r8	2/2	Move byte register to r/m byte
89 /r	MOV r/m16,r16	2/2	Move word register to r/m word
89 /r	MOV r/m32,r32	2/2	Move dword register to r/m dword
（以下略）			

2. shell 命令

（1）完成 PA1 的内容之后，nemu 目录下的所有.c 和.h 和文件总共有多少行代码？

使用的命令：

```
cat `find nemu -name '*.[ch]'` | wc -l
```

执行结果：

```
5845
```

命令解释：

通过 find nemu -name '*.[ch]'列出 nemu 目录下所有.c 与.h 文件的路径与文件名；然后把结果作为参数调用 cat 命令，cat 命令会把这些文件的内容拼接到一起，输出到通过管道连接的 wc -l 命令；wc -l 命令会统计输入数据的行数，进而得到所需要的统计结果。

（2）除去空行之外，nemu 目录下的所有.c 和.h 文件总共有多少行代码？

使用的命令：

```
cat `find nemu -name '*.[ch]'` | sed '/^\s*$/d' | wc -l
```

执行结果：

```
4712
```

命令解释：

命令的大部分与之前一样，不同之处在于用管道与 sed 命令进行了进一步处理：命令 sed '/^\s*$/d'可以删除空行和仅含有空白符的行，这样就可以产生所需要的统计结果。

3. 使用 man

（1）请解释 gcc 中的 -Wall 和 -Werror 有什么作用？

使用 man gcc 命令查阅手册，搜索关键字 Wall 得到如下结果，可以看出 -Wall 开关的作用是：打开一些警告选项，提醒程序员代码中可疑的代码结构（包括使用宏产生的）。

-Wall
This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings described in C++ Dialect Options and Objective-C and Objective-C++ Dialect Options.

使用与之前相同的方法，在 gcc 的手册页中查询到如下结果，可以看出 -Wall 开关的作用是：使所有警告变为错误，拒绝编译会产生警告的代码。

```
-Werror
   Make all warnings into hard errors.  Source code which
triggers warnings will be rejected.
```

（2）为什么要使用 -Wall 和 -Werror？

使用 -Wall 选项可以帮助程序员避免一些低级错误（比如：误把等于运算符"=="打成赋值运算符"="、误把逻辑或运算符"||"打成或运算符"|"、弄错运算符优先级导致运算结果错误等）。使用 -Werror 可以在遇到警告时立即终止编译过程，强制程序员对代码进行修改，防止警告信息被无意忽略。

三、实验心得和想法

1．做笔记很重要（笔记存储在/nemu/ZBYnotes 目录下）

当我刚刚拿到 NEMU 的框架代码时，我完全不了解整个代码的结构，读起来很费劲。有时候读了一些代码，第二天就忘记了那段代码在哪里、是做什么的，这样我基本上等于白读了这些代码。有句话叫"好记性不如烂笔头"，于是我决定还是做笔记比较好：把每天读的代码、分析出的结果、遇到的问题写在笔记里面，这样每次只要看一下之前的笔记，就很容易想起来之前分析得到的结果，不必要每次都重新读代码，节省了时间。当代码的结构在脑中建立起来后，我就把每天做了些什么、要重点关注的地方、从哪里查找到的资料写进笔记中。这样每次只要查一下笔记里面的内容，就能迅速回想起当时思考问题、解决问题的过程，节约了时间。

2．辩证看待 KISS 原则

在"表达式求值"一节中提到了 KISS 原则，但我并不认为这条"黄金法则"在各处都适用。我认为在设计整个系统的过程中，就一定要追求完美。一个好的系统设计，可以事半功倍；而一个烂的系统设计，很可能导致事倍功半。例如在设计"表达式求值"的过程中，我们一开始就设计好了这个先"用词法分析的方法把输入字符串分解为一个一个的 Token"然后再"递归处理 Token 并计算结果"的实现方式。如果一味地使用 KISS 原则，简化系统设计部分，那么很可能写出了一份"OI 题目式的表达式求值"——边处理字符串边计算的实现方式。显然前者比后者维护性更好，代码更简洁清晰。假设已经按照后者写完了简单的"算数表达式求值"，但是为了实现更多运算符，要把实现方式从后者改为前者，这时会发现：改动一开始的设计不仅麻烦，而且还很难实现代码重用，相当于之前写的代码基本白写了。因此，我认为运用 KISS 原则时，一定要思考好它的利与弊，不能盲目地使用。

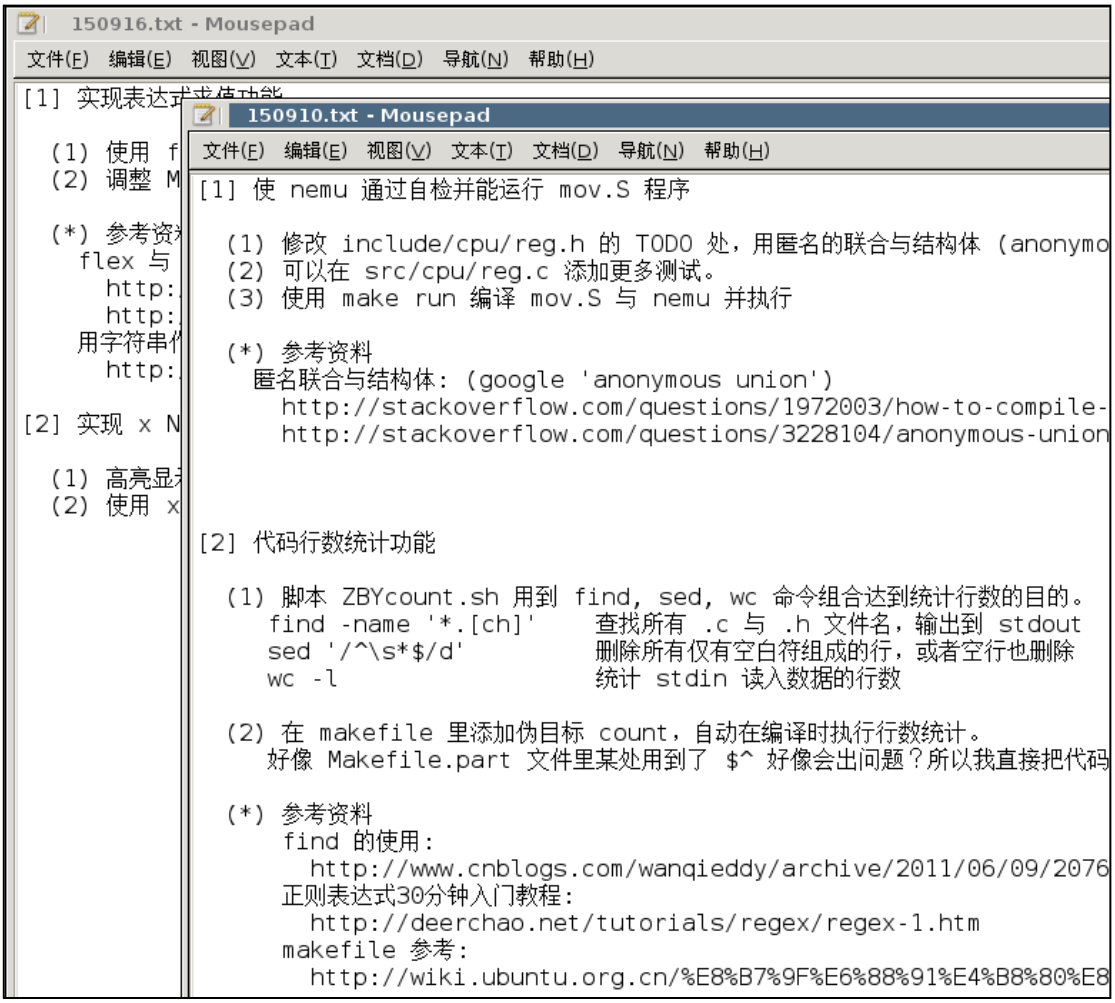
3．自己对自己负责

在编写代码过程中，很可能产生"好烦啊，随便写写算了"这样的想法，然而只要思考一下这样做的利与弊，就会打消掉这样的想法。自己写出来的代码自己负责，出来混总是要还的。如果现在偷懒，没准就在代码里面留下了 Bug，它很可能在 PA2、PA3、PA4 中跳出来，耗掉好几个小时来调试这个简单问题。如果现在多测试几遍，确认代码无 Bug，很可能就可以避免在之后的开发中遇到棘手的问题。另外，多实现一些实用的功能，也是在方便自己。例如我实现了用分号连接多个命令的功能，这样就可以直接用 si; i r 指令边单步执行边打印寄存器内容，省去了每次都要重新输入 si 与 i r 指令的麻烦。另外，我还实现了用红色显示寄存器新旧值之间的差异的

功能，这样就可以在单步执行时，一眼就可以看出哪个寄存器、哪些位的值发生了改变。多实现一些有用的功能，看似增加了工作量，实际上是方便了自己之后的开发，也是一种自己对自己负责的方式。

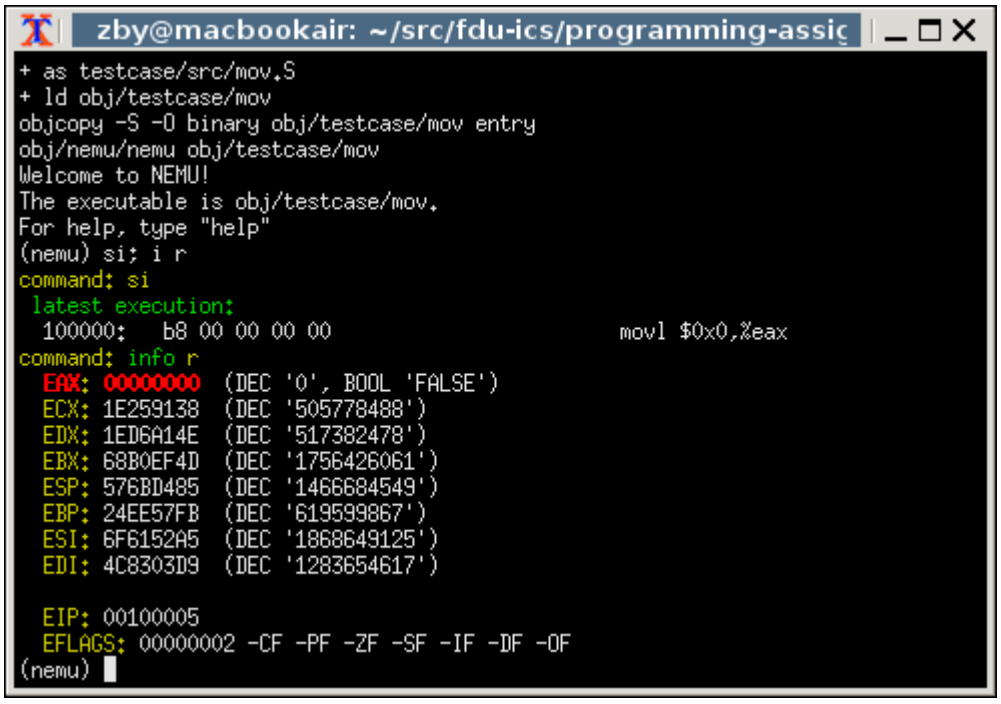
四、附录

1. 部分笔记的截图

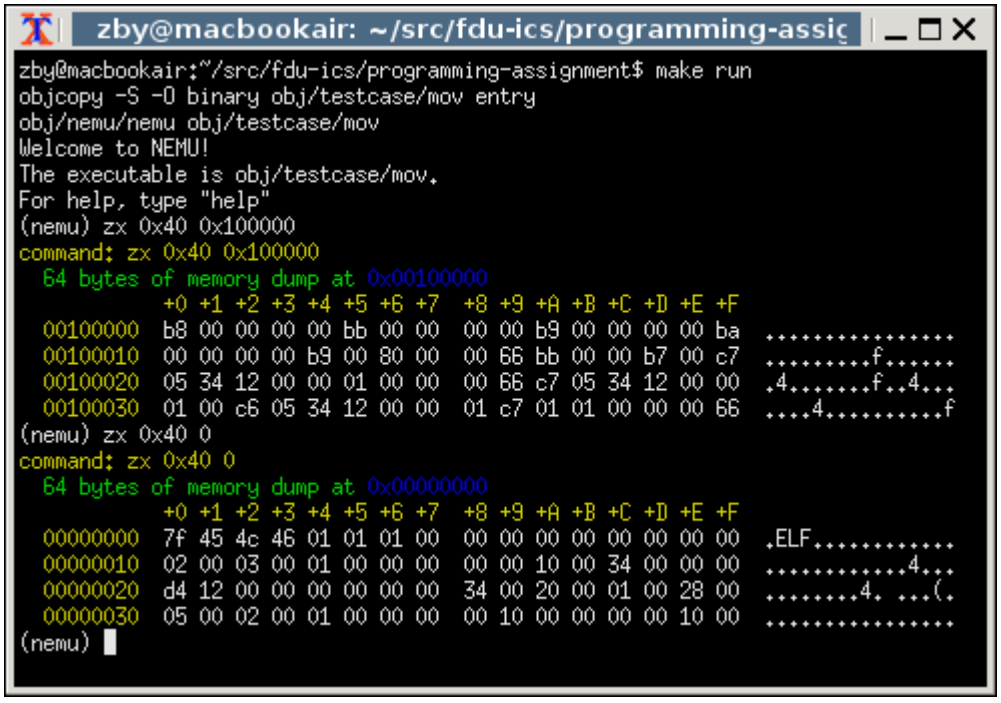


2. 实现的额外功能。

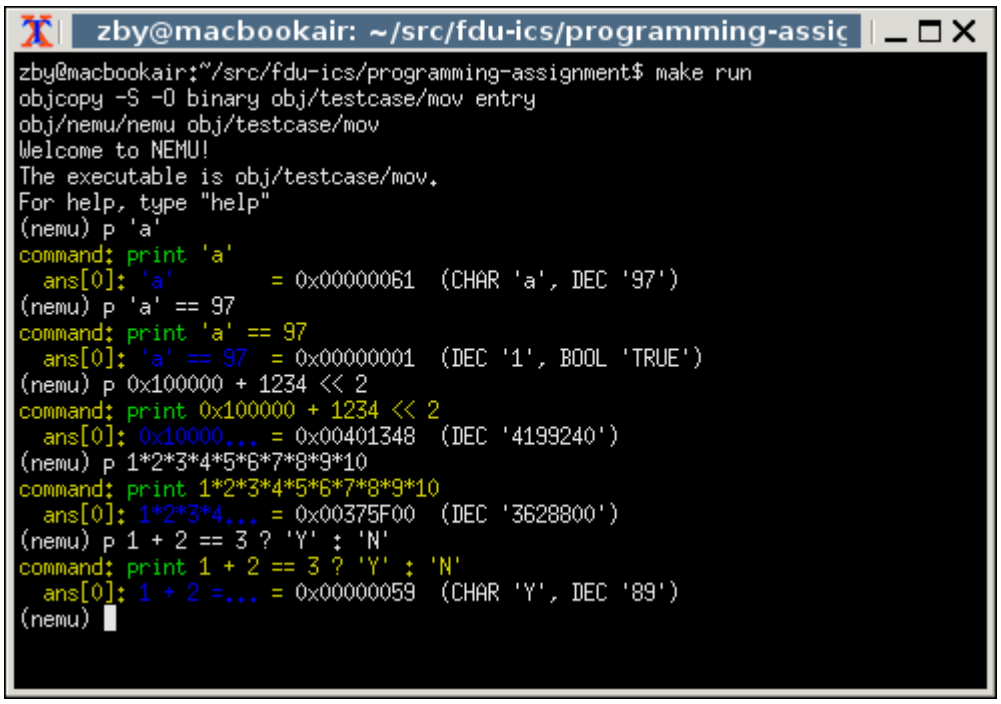
（1）用 `si; i r` 实现单步执行并高亮显示改变的寄存器



（2）用 `zx` 命令显示内存内容



（3）对于数值型数据，可以猜测并显示数据的意义



（4）p 命令支持逗号分隔的多个表达式同时求值