你需要在实验报告中回答下列问题:

> 送分题 我选择的ISA是

x86

> 程序是个状态机 画出计算 `1+2+...+100` 的程序的状态机, 具体请参考 **这里**.

- 前两轮：0:0:0->1:0:0->2:0:1->3:1:1->4:1:1->2:1:2->3:3:2->4:3:2
- 后两轮：2:4851:99->3:4950:99->4:4950:99->2:4950:100->3:5050:100->5:5050:100

> 理解基础设施 我们通过一些简单的计算来体会简易调试器的作用. 首先作以下假设:
>
> - 假设你需要编译500次NEMU才能完成PA.
> - 假设这500次编译当中, 有90%的次数是用于调试.
> - 假设你没有实现简易调试器, 只能通过GDB对运行在NEMU上的客户程序进行调试. 在每一次调试中, 由于GDB不能直接观测客户程序, 你需要花费30秒的时间来从GDB中获取并分析一个信息.
> - 假设你需要获取并分析20个信息才能排除一个bug.
>
>   那么这个学期下来, 你将会在调试上花费多少时间?
>
>   由于简易调试器可以直接观测客户程序, 假设通过简易调试器只需要花费10秒的时间从中获取并分析相同的信息. 那么这个学期下来, 简易调试器可以帮助你节省多少调试的时间?
>
>   事实上, 这些数字也许还是有点乐观, 例如就算使用GDB来直接调试客户程序, 这些数字假设你能通过10分钟的时间排除一个bug. 如果实际上你需要在调试过程中获取并分析更多的信息, 简易调试器这一基础设施能带来的好处就更大.

我将花费$500 * 0.9 * 30 * 20 = 270000s = 225h$

我将减少花费$225 * 2/3 = 150h$

> RTFM
>
> 理解了科学查阅手册的方法之后, 请你尝试在你选择的ISA手册中查阅以下问题所在的位置, 把需要阅读的范围写到你的实验报告里面:
>
> x86
> EFLAGS寄存器中的CF位是什么意思?
> ModR/M字节是什么?
> mov指令的具体格式是怎么样的?

- CF位表示无符号数是否溢出，即是否发生了借位。 --Appendix C
- This byte, called the ModR/M byte, specifies the address form to be used. --17.2.1
- 17- **MOV Move Data**; **MOV Move to/from Special Registers**

> shell命令 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? 你是使用什么命令得到这个结果的? 和框架代码相比, 你在PA1中编写了多少行代码? (Hint: 目前 `pa0` 分支中记录的正好是做PA1之前的状态, 思考一下应该如何回到"过去"?) 你可以把这条命令写入 `Makefile` 中, 随着实验进度的推进, 你可以很方便地统计工程的代码行数, 例如敲入 `make count` 就会自动运行统计代码行数的命令. 再来个难一点的, 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?

```
# non-empty
find /home/qwe/ics2020/nemu |sed -n '/\.c$/p;/\.h$/p' |xargs cat|grep -v ^$|wc -l
# include empty lines
find /home/qwe/ics2020/nemu |sed -n '/\.c$/p;/\.h$/p' |xargs cat|wc -l
```

nemu中的.c和.h文件共有5749行代码。我使用shell脚本：./count

切换回pa0，我多写了5749-5153= 596行代码

非空代码有4760行。

> RTFM
>
> 打开工程目录下的 `Makefile` 文件, 你会在 `CFLAGS` 变量中看到gcc的一些编译选项. 请解释gcc中的 `-Wall` 和 `-Werror` 有什么作用? 为什么要使用 `-Wall` 和 `-Werror` ?

-Werror: 把所有的warning当做error提出

-Wall: 这将启用所有被认为可疑构造的警告，并且它们是很容易避免的，即稍作修改就可以使警告消除。（即使与宏结合使用）