

**实验报告**



**题目： 熟悉Linux系统及其相关软件环境**

**班 级： 2020211305**

**学 号： 2020211346**

**姓 名： 倪玮昊**

**学 院： 计算机学院**

**2021年 10 月 30 日**

1. 实验目的
2. 熟悉linux操作的基本操作；
3. 掌握gcc编译方法；
4. 掌握gdb的调试工具使用；
5. 掌握objdump反汇编工具使用；
6. 熟悉理解反汇编程序（对照源程序与objdump生成的汇编程序）。

二、实验环境（10分）

简述使用的工具

1. shell
2. Linux
3. Gcc编译器
4. GDB调试工具
5. Objdump命令反汇编
6. vi编译器

三、实验概况

简述实验内容和基本设想

实验内容一（15分）

在linux环境下，编辑课件中源程序（注意程序的完整性）（包含源程序的开发环境截图）

采用gcc编译该程序（要求分别采用-o和-O参数，并比较两者性能，编译指令截图），

采用gdb进行调试，让程序运行到for函数语句（调试截图），

运用objdump工具生成汇编程序（给出main函数的汇编程序截图）

#include<stdio.h>

int main(void)

{

double counter;

double result;

double temp;

for(counter=0;counter<2000.0\*2000.0\*2000.0/20.0+2020;

counter+=(5-1)/4){

temp=counter/1979;

result=counter;

}

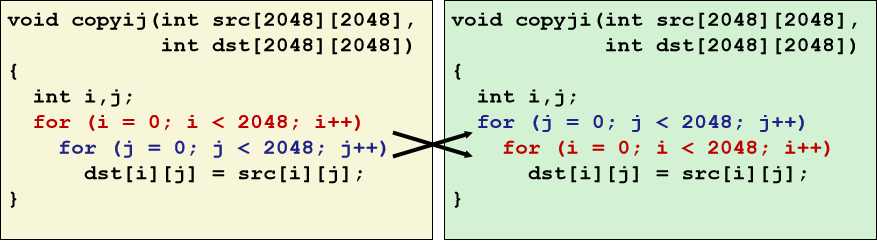
printf(Result is%lf\\n,result);

return 0；

}

实验内容二（15分）

在linux环境下，分别打印输出如下算法所需时间



分别设置不同优化参数，给出运行时间

实验内容三（30分）

现有int型数组a[i]=i-50,b[i]=i+y，其中y取自于学生本人学号2019211x\*y的个位。登录bupt1服务器，在linux环境下使用vi编辑器编写C语言源程序，完成数组a+b的功能，规定数组长度为100，函数名为madd（），数组a，b均定义在函数内，采用gcc编译该程序（不使用优化选项），

使用objdump工具生成汇编程序，找到madd函数的汇编程序，给出截图；

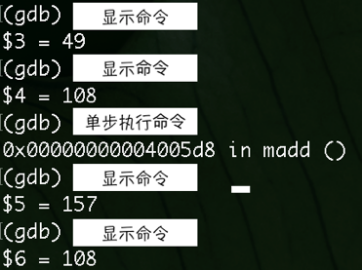
用gdb进行调试，练习如下gdb命令，给出截图；

gdb、file、kill、quit、break、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch

找到a[i]+b[i]对应的汇编指令，指出a[i]和b[i]位于哪个寄存器中，给出截图；

使用单步指令及gdb相关命令，显示a[xy]+b[xy]对应的汇编指令执行前后操作数寄存器十进制和十六进制的值，其中x，y取自于学生本人学号2019211x\*y的百位和个位。

学号2019211999，a[99]+b[99]单步执行前后的参考截图如下（实际命令未显示出）：



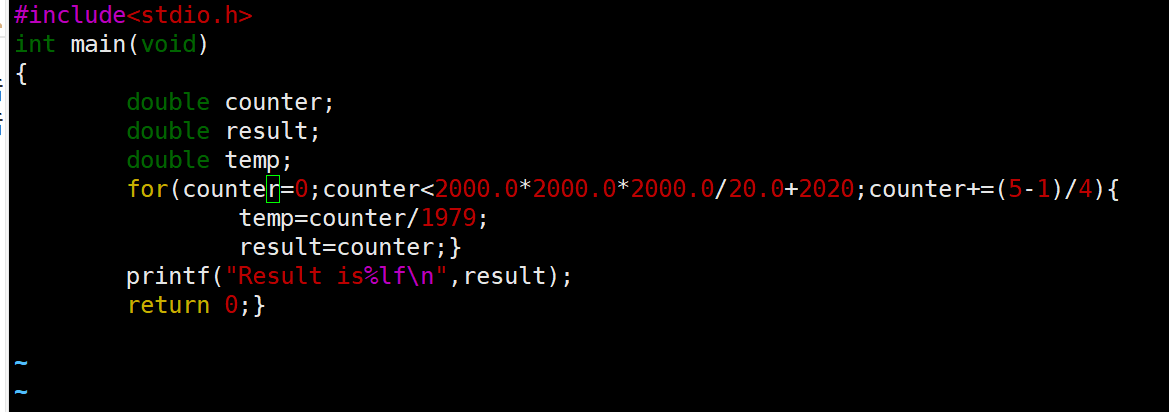
实验内容四（加分项，20分）

任选高复杂度算法（具体算法自选，类型分为高计算量类型和高内存需求类型2类算法），通过设置不同优化参数，分析算法的运行效率

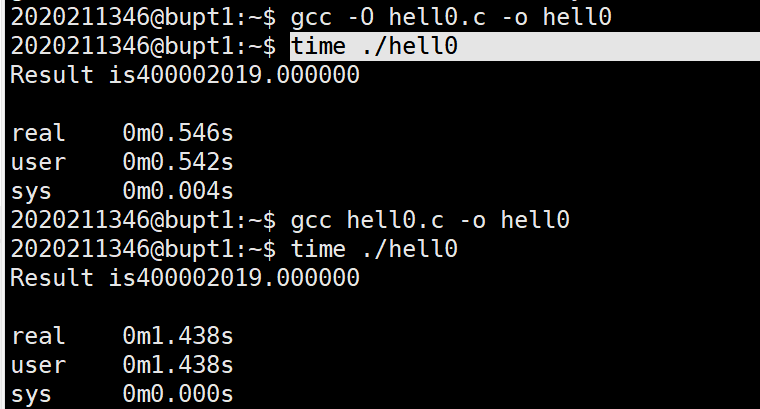
四、实验步骤（60-80分）

实验1：

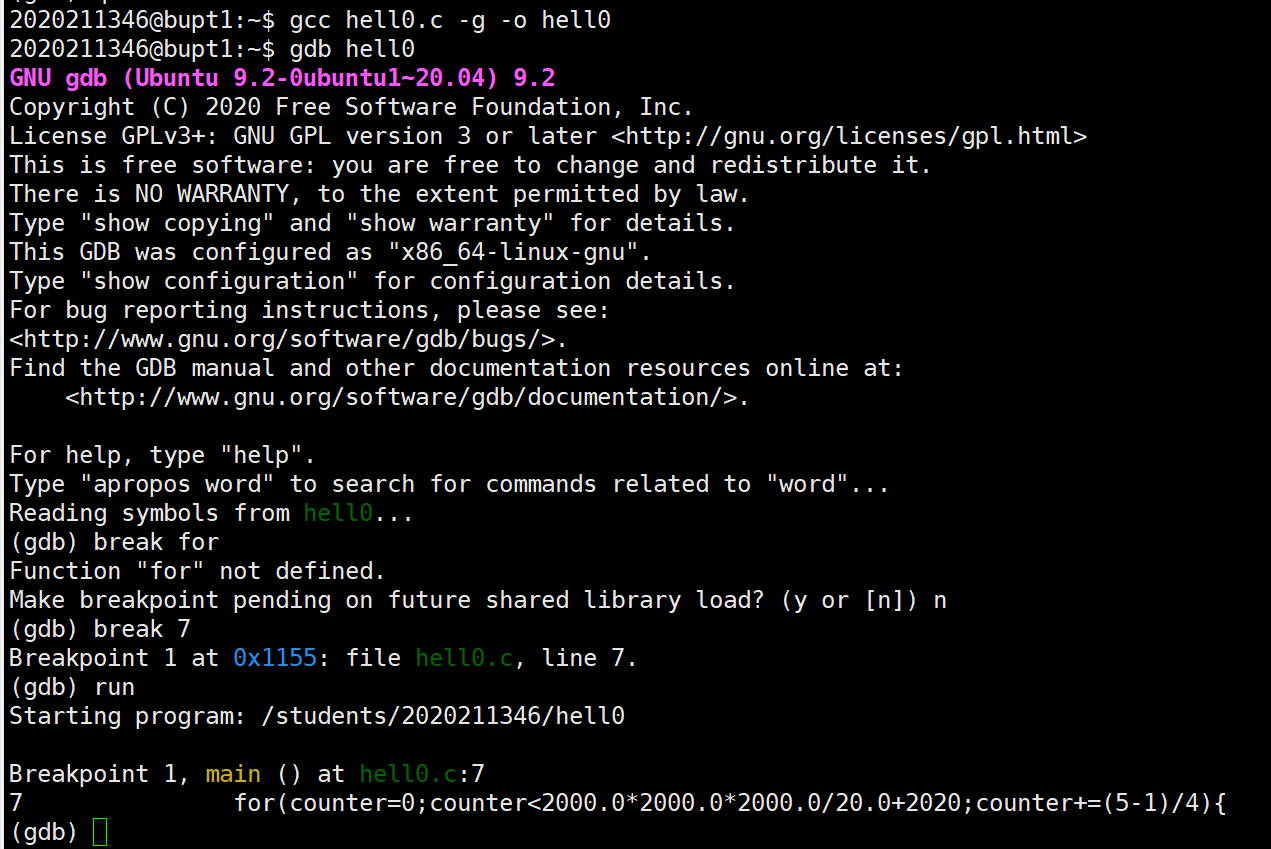
1. 输入 vi hell0.c 打开空白文档，输入代码，按esc键切换为命令模式，输入：w来储存内容，再输入：x关闭vi编译器。输入代码如下图：



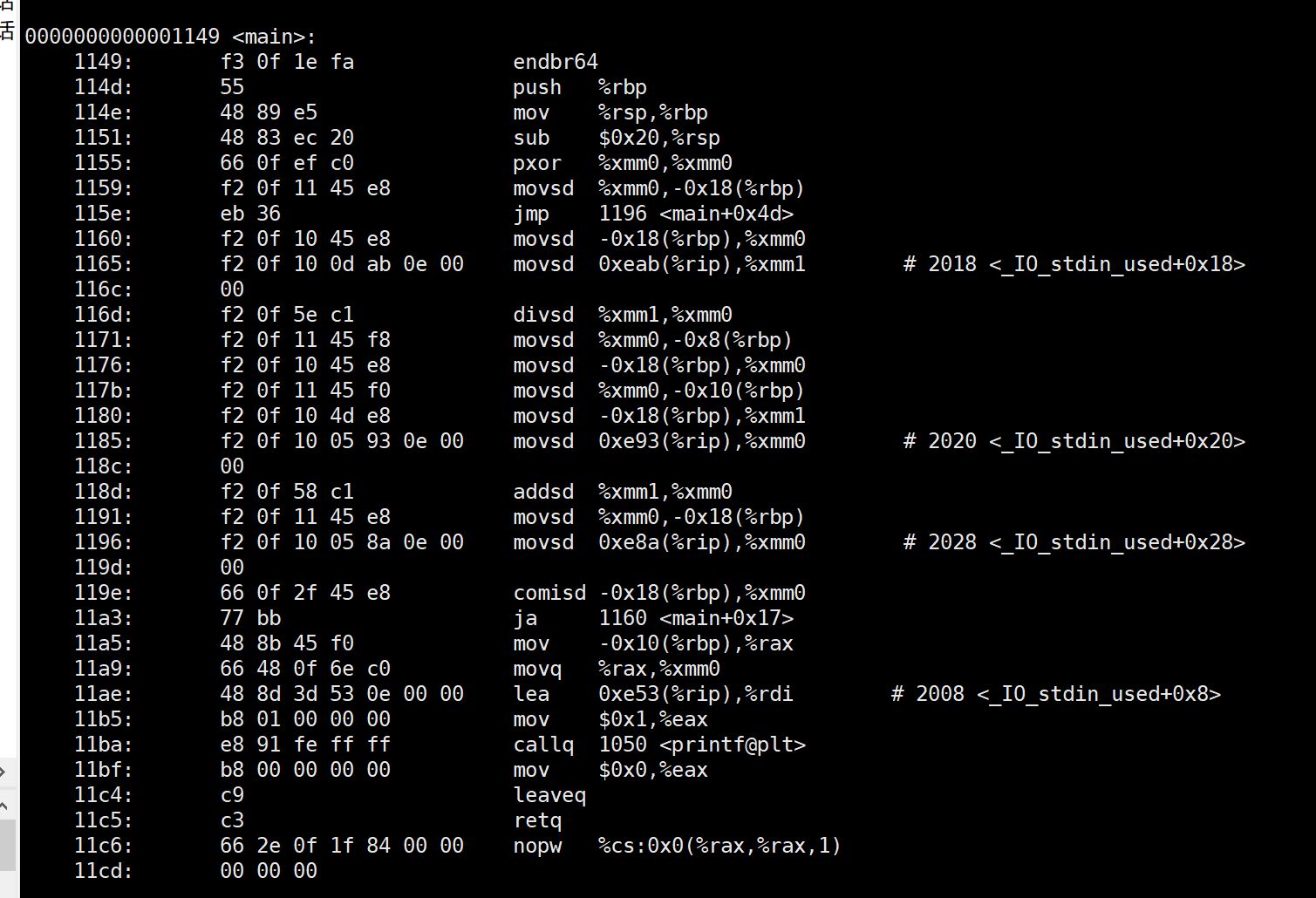
1. 输入gcc -O1 hell0.c -O hell0来计算使用O时间长度。输入gcc hell0.c -o hell0来计算使用o时间长度。



1. 输入指令 gcc hell0.c -g -o hell0使文件具有debug条件，输入gdb hell0进入gdb，由源代码可知for循环再第7行，输入命令break 7 再输入run，函数运行到for循环即停止。



1. 输入gcc hell0.C -g -o hell0进行编译，完成后输入objdump -d hell0生成汇编程序



实验二：

打开vi编辑器，分别输入两组函数，如图：





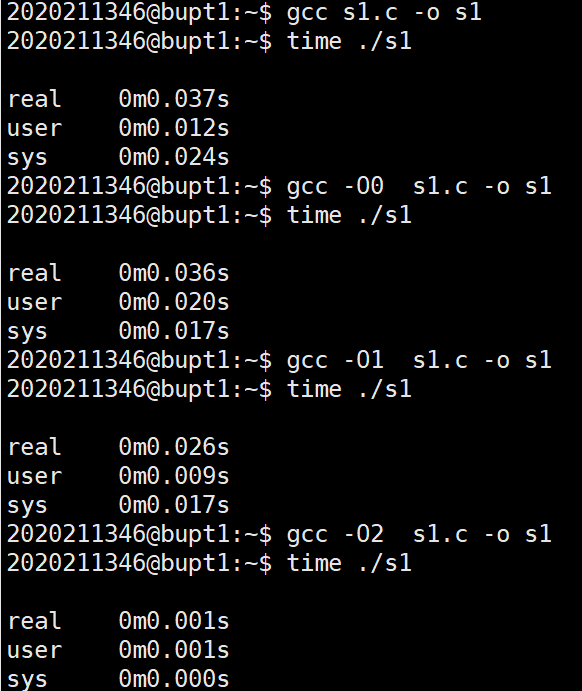
分别输入gcc -O s1.c -O s1和time ./s1

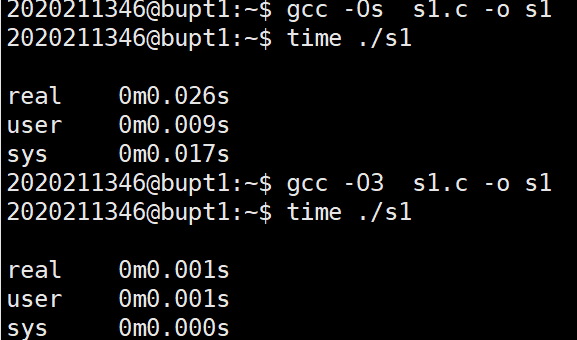
gcc -O1 s1.c -O s1和time ./s1

gcc -O2 s1.c -O s1和time ./s1

gcc -O3 s1.c -O s1和time ./s1

计算四种优化的运行时间





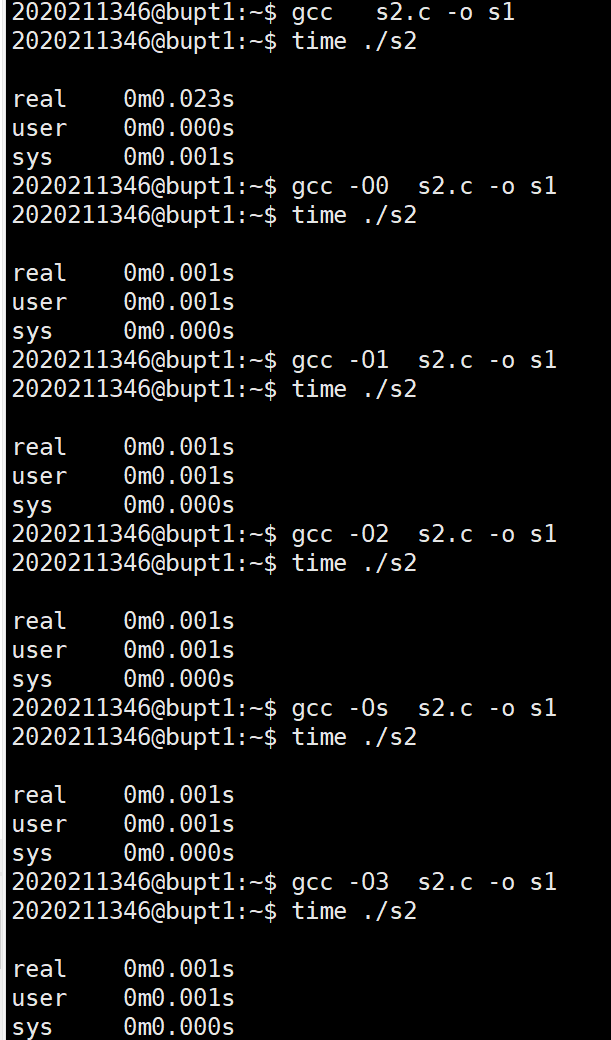
分别输入gcc -O s2.c -O s1和time ./s2

gcc -O1 s2.c -O s2和time ./s2

gcc -O2 s2.c -O s2和time ./s2

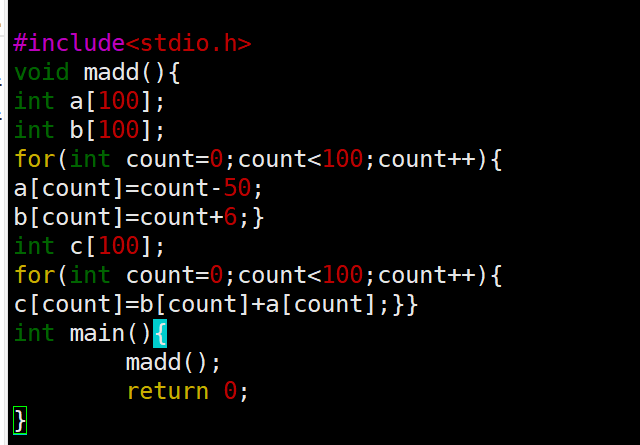
gcc -O3 s2.c -O s2和time ./s2

计算四种优化的运行时间

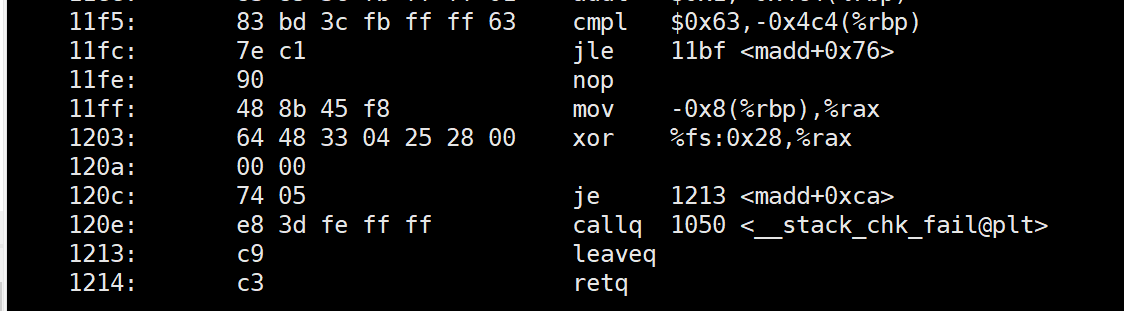
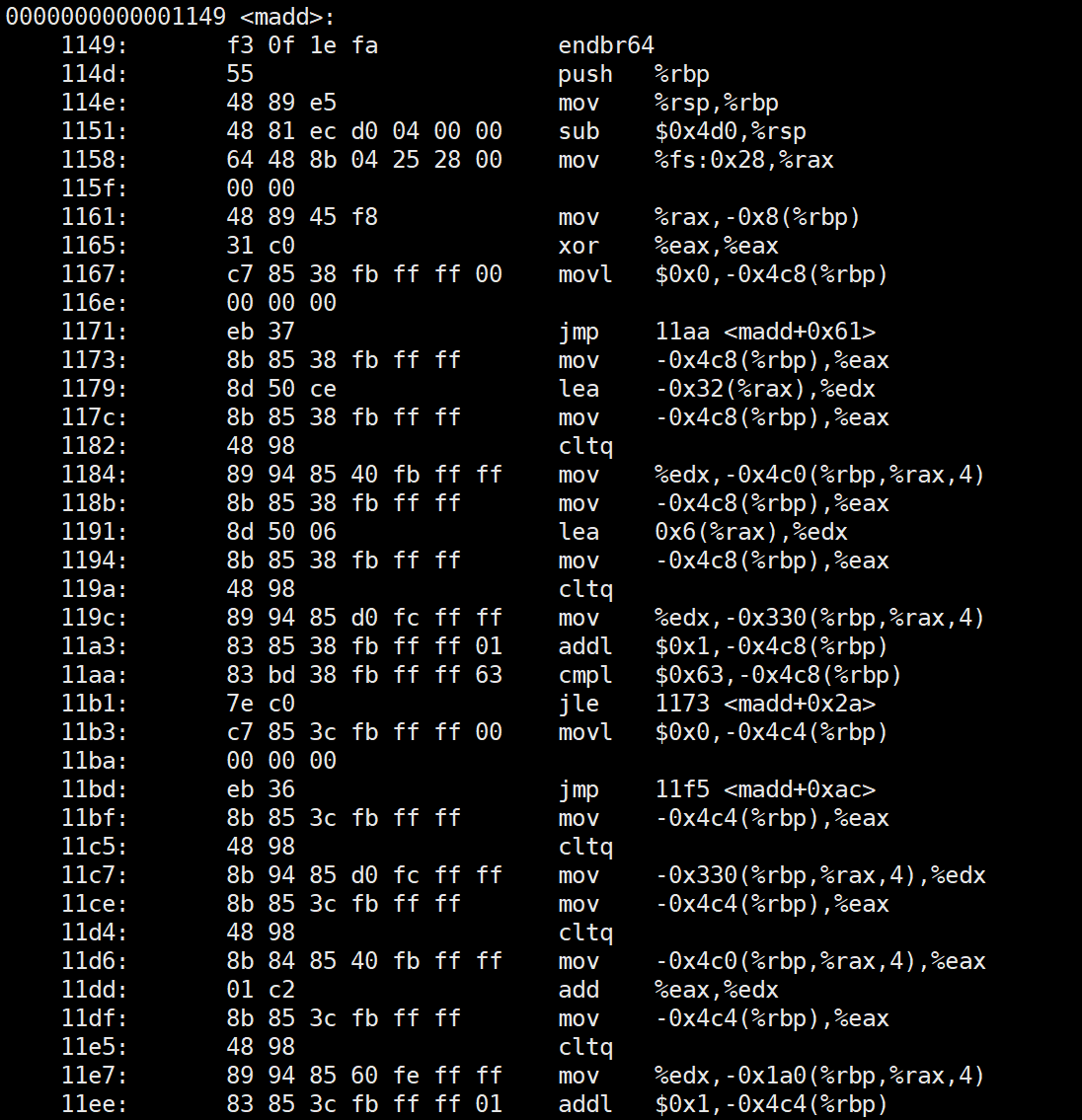


实验3：

1. 输入 vi madd.c 打开空白文档，输入代码，按esc键切换为命令模式，输入：w来储存内容，再输入：x关闭vi编译器。输入代码如下图：



2.输入gcc madd.c -g -o madd进行编译，完成后输入objdump -d madd生成汇编程序



1. 使用gdb调试：

图一：

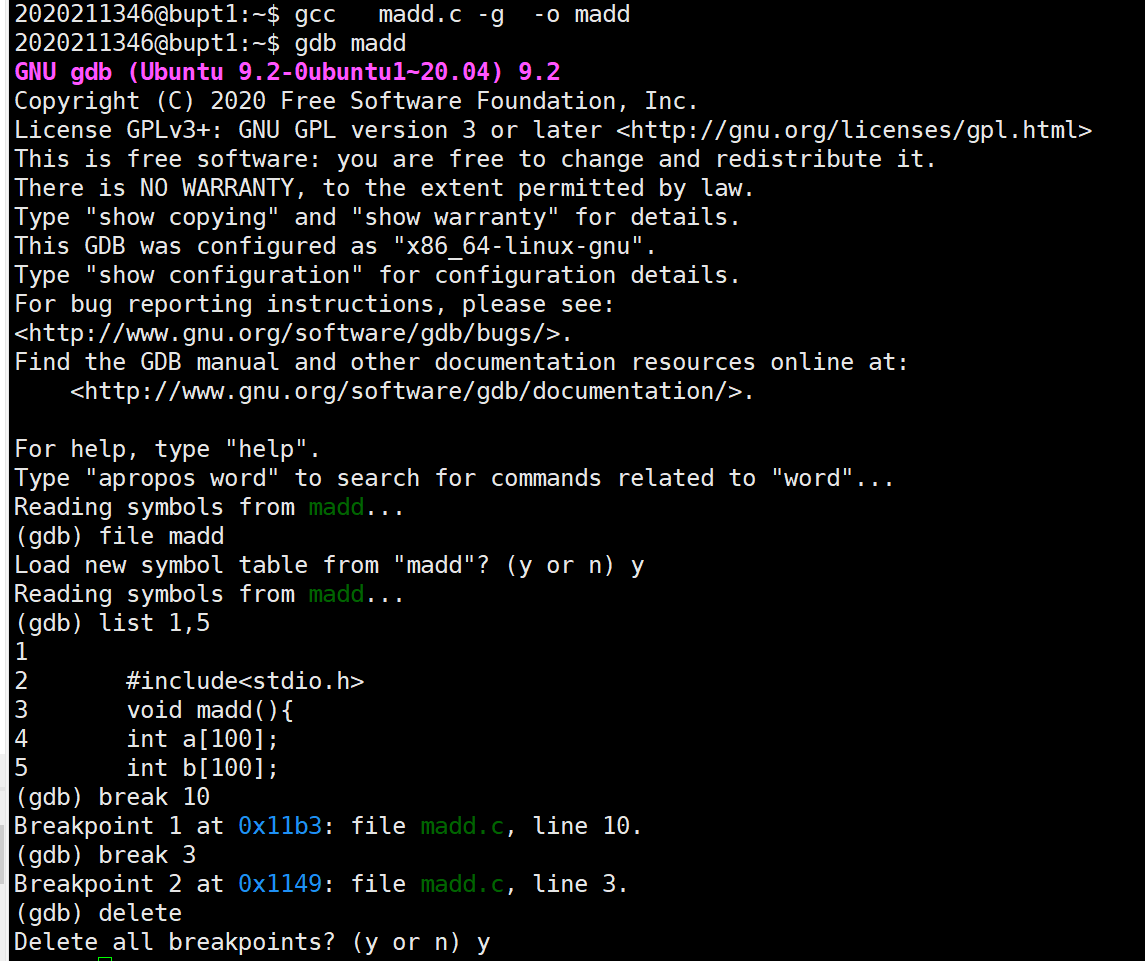
gdb：进入gdb进行调试

file：加载被调试的可执行程序文件

list: 输入list a b打印源代码a行到b行的代码

break：设置断点

delete：删除全部断点



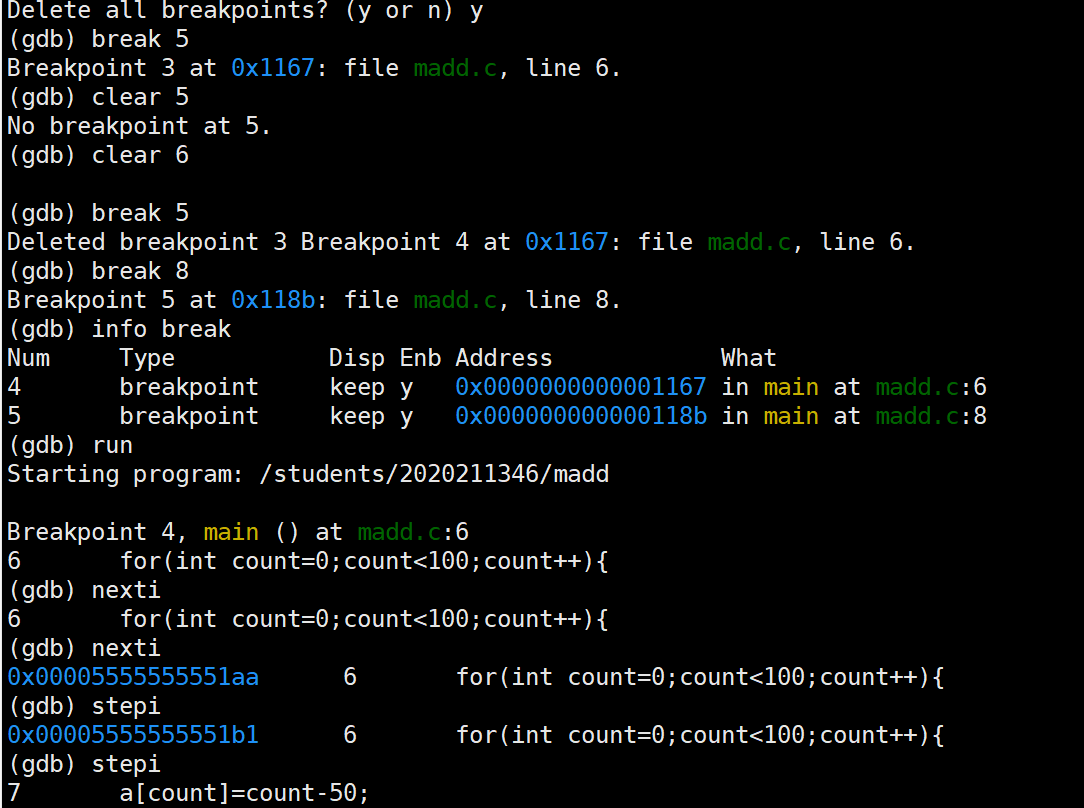
clear：删除某一行的断点

info break：查看设置的所有断点

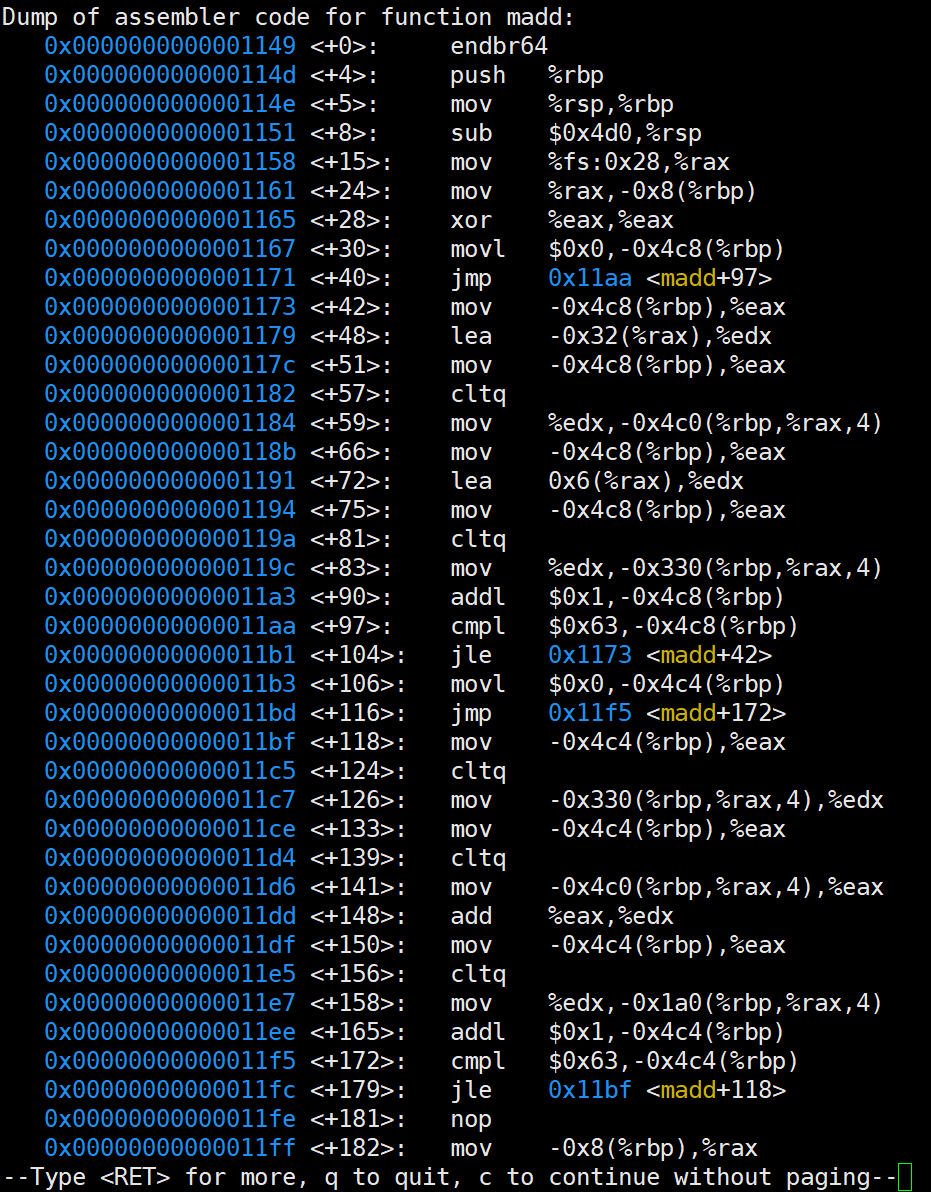
run：运行程序

nexti：进入下一个机器指令，遇到子函数不进入，直接执行整个函数

stepi：进入下一个机器指令，遇到子函数仍然进入并单步执行



disassemble：让汇编语言按照inter格式或者是AT&T的格式显示



print：打印某个变量的值

info reg：查看寄存器使用情况

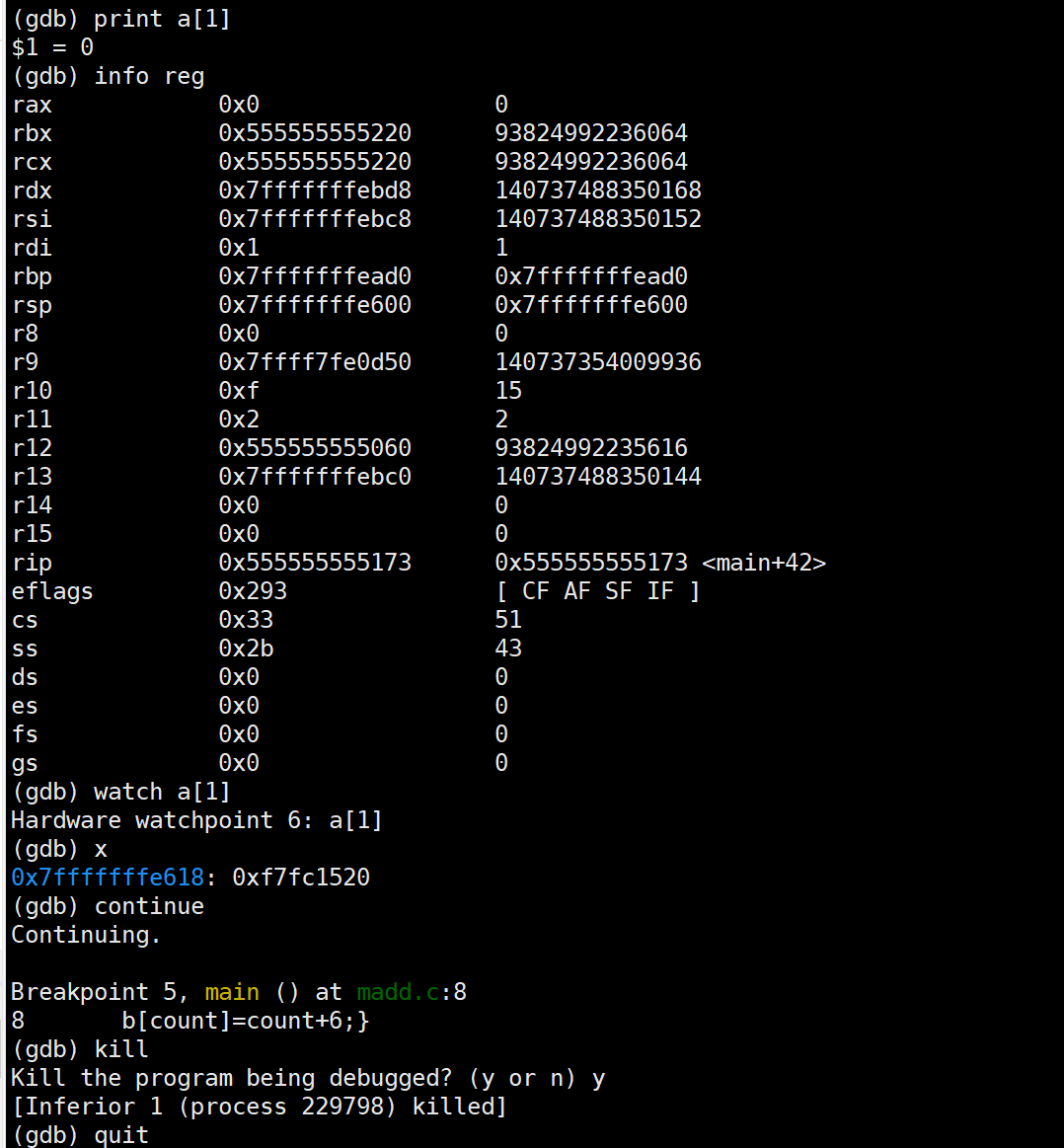
watch：监控某个变量的变化

x：显示从此向后n个单元的内存内容

continue：从断点继续执行

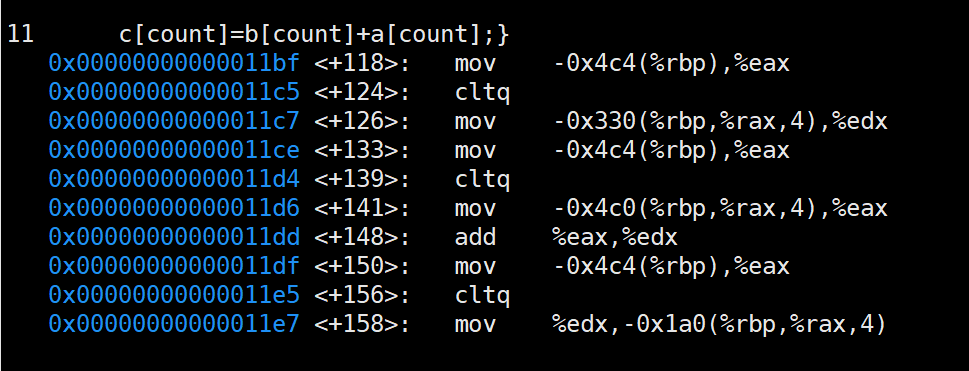
kill：结束当前程序

quit：退出gdb工具

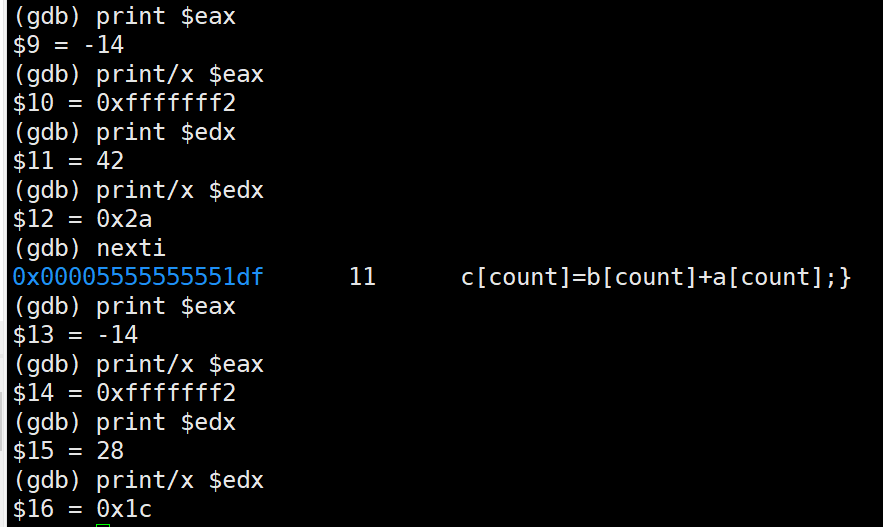


1. 如下图所示，有指令 add %eax %edx 故二者为a[]和b[]的寄存器

且a[]对应%eax b[]对应

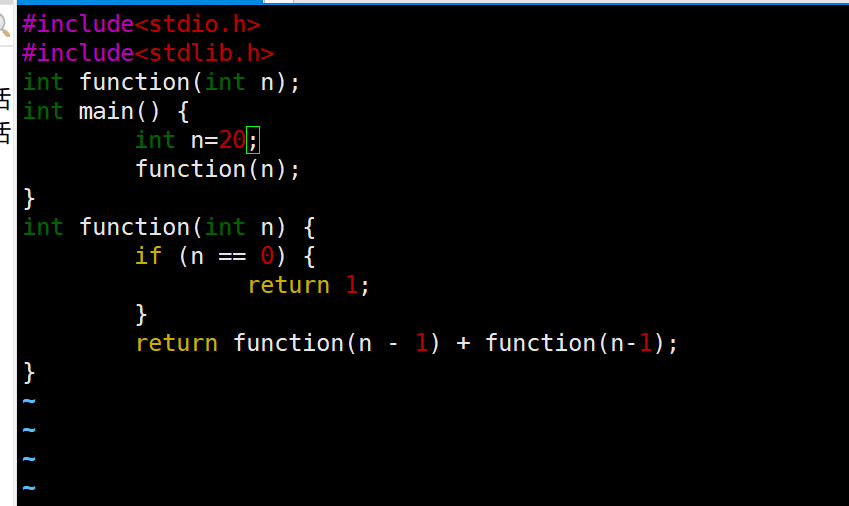
于%edx.

1. 在for循环处设置断点，输入run，运行到断点时输入next73，正好运行到count=36，再使用nexti，一步一步运行，同时print $eax和$edx，直到两个寄存器寄存的是a[]和b[]两个操作数，如图是操作前后的十进制和十六进制变化。

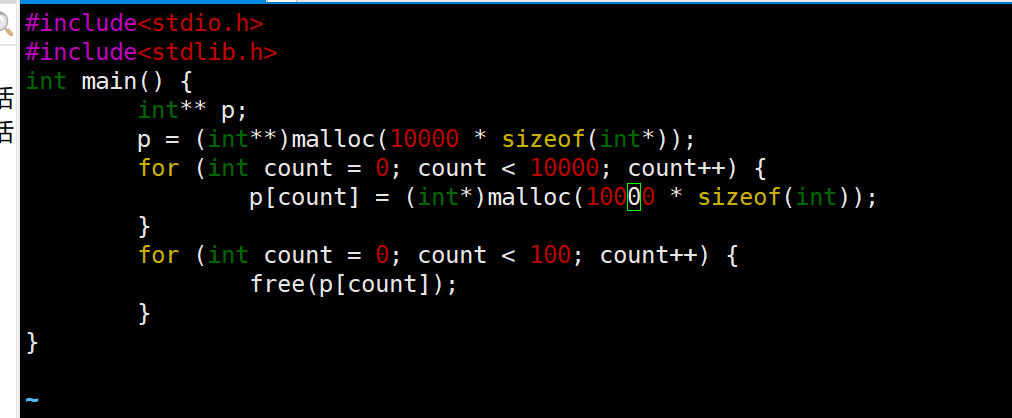


实验4：

分别编写高计算量算法 hightime.c(时间复杂度为2^n)：

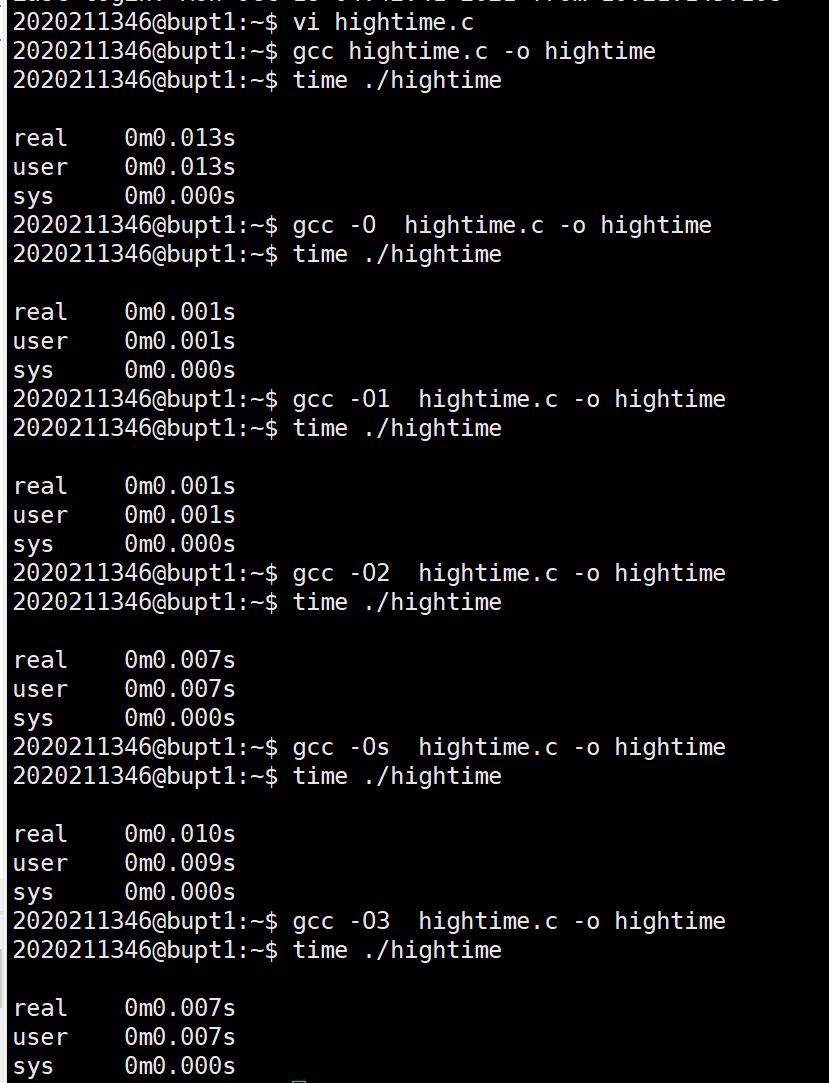


和高内存需求算法highspace，c（空间复杂度为n^n)：

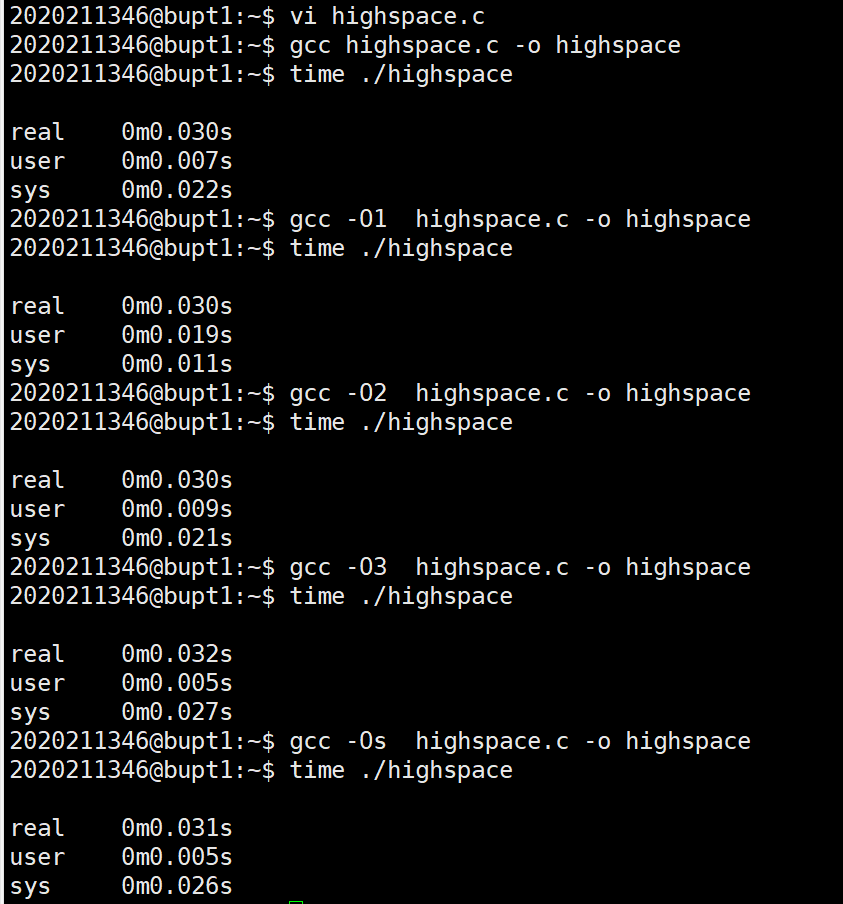


分别用不同的优化程序进行优化得出运行时间从来分析算法效率

hightime.c：



highspace.c:



1. 实验分析（20分）

实验1：

1.首先我的目标是写一个程序，在这里需要建立一个文本并且输入我的程序内容，输入完成以后使用gcc进行编译，成功得到了所输入的程序

2.在编译hell0.c成功后，我分别用-o和-O进行优化，再使用时间函数计算时间长度，可见-o组在时间上明显减少。

3.在gcc中使用-g指令使hell0具有编译条件，在进入gdb对程序进行调试，观察源代码得知for在第7行，输入break 7，函数在for处中止。

4.在gcc中完成程序编译后使用objdumo生成汇编程序

实验2：

使用vi编辑器输入s1.c和s2.c两组程序main函数改变版,编译完两组程序后分别使用-O，-O1，-O2，-O3来对程序进行优化，并使用时间函数来计算程序运行时间，可以看出第一个程序效率高于第二个程序，且优化效率O3>O2>Os>O1>O0

实验3：

1. 用vi编辑器打开空白文档输入madd.c，编译后用objdump -d madd生成汇编程序
2. 在gdb中使用各种命令进行测试，各种功能如下

gdb：进入gdb进行调试

file：加载被调试的可执行程序文件

list: 输入list a b打印源代码a行到b行的代码

break：设置断点

delete：删除全部断点

clear：删除某一行的断点

info break：查看设置的所有断点

run：运行程序

nexti：进入下一个机器指令，遇到子函数不进入，直接执行整个函数

stepi：进入下一个机器指令，遇到子函数仍然进入并单步执行

disassemble：让汇编语言按照inter格式或者是AT&T的格式显示

print：打印某个变量的值

info reg：查看寄存器使用情况

watch：监控某个变量的变化

x：显示从此向后n个单元的内存内容

continue：从断点继续执行

kill：结束当前程序

quit：退出gdb工具

1. 通过使用 disassemble/m madd显示每一步的机器指令，找到c[count]=b[count]+a[count] 步骤下的机器指令 有指令add %eax%edx可知a[]对应寄存器%eax，b[]对应寄存器%edx。

4.本人学号为2020211346 故xy=36

在for循环处设置断点，输入run，运行到断点时输入next72，正好运行到count=36，再使用nexti，一步一步运行，同时print $eax和$edx，直到两个寄存器寄存的是a[]和b[]两个操作数，分别使用print madd和print/x madd来显示十进制结果和十六进制结果

实验4：编写高内存需求算法highspace，c（空间复杂度为n^n)和高计算需求算法highspace.c(刚开始我把两个程序的n都设置为100，但前者运行太快看不出优化效果，后者运行太慢，故对n的值进行了修改）

从运行时间可以看出gcc的三个优化对高内存需求运算的算法效率优化并不明显。

从运行时间可以看出gcc的优化对高计算需求算法运行效率提升明显且并未按优化级别升高而更快，O1的优化效果最好，O2和O3持平，Os效果最差但仍然好于无优化。

六、实验总结（10分）

1.在输入代码时要谨慎，看准了文本，不要漏变量也不要输错符号，因为在linux下的gcc编译不像我们直接用dev c++或者vs2019这样检查很方便，刚输完是不会提示你输入代码是否正确，需要退出编辑器再编译再进入编辑器进行修改。

2.在使用gdb前需要输入指令-o来使文件具有debug条件 否则无法进行调试。

3.寄存器里寄存的变量会发生变化，而且可能在一行代码里变化多次，所以要在某行里慢慢用nexti来进入一行行机器指令来找到我们要求的操作数。

4.gcc和gdb的指令很多，需要通过检索和学习才能熟悉，在有困难的时候求助于百度和csdn是个好选择，有利于提高我们自学能力。

5.汇编代码的阅读难度很大，但有利于我真正的了解编译的原理和过程，在一次调试中我也明白了各个寄存器的值是怎么变化的（在这里确实卡了很久）。

6.在gcc的优化中，不一定优化级别越高运行越快，往往和函数构造有一定关系。

7.编写高内存需求算法和高计算需求算法时我把两个程序的n都设置为100，但前者运行太快看不出优化效果，后者运行太慢，故对n的值进行了修改。

七、诚信声明（不签扣10分）

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

1. 简单描述交流内容：未咨询任何人
2. 此外，我还参考了以下资料：

网址：

1. https://blog.csdn.net/bible\_reader/article/details/80210880
2. <https://blog.csdn.net/leikezhu1981/article/details/44831999>
3. <https://www.cnblogs.com/liwei0526vip/p/14761934.html>
4. https://blog.csdn.net/wuxing26jiayou/article/details/96132721

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。



（签名）

