

**实验报告**



**题目： Linux环境和GCC工具链**

**班 级： 2023211307**

**学 号： 2023211075**

**姓 名： 魏生辉**

**学 院： 计算机学院**

**2024 年 10 月 15 日**

一**、实验目的**

1. 熟悉linux系统的常用命令；
2. 掌握gcc编译器的使用方法；
3. 掌握gdb的调试工具使用；
4. 掌握objdump反汇编工具使用；
5. 理解反汇编程序（对照源程序与objdump生成的汇编程序）。
6. **实验环境+-**

列举所使用的软件工具

**1**. 远程登陆工具（服务器：10.120.11.12）

**2**. Linux:5.4.0-156-generic x86\_64

**3.** Gcc编译器:Ubuntu 7.5.0-6ubuntu2

**4**. GDB调试工具：GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2

**5.** GNU objdump (GNU Binutils for Ubuntu) 2.34

**三、实验内容**

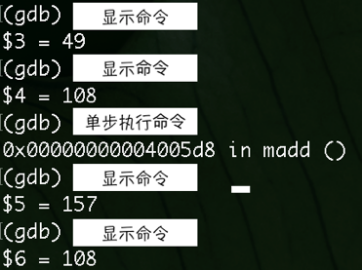
现有两个int型数组a[i]=i-50，b[i]=i+y，其中y取自于学生本人学号2022211x\*y的个位。登录bupt1服务器，在linux环境下使用vi编辑器编写C语言源程序，完成数组a ← a+b的功能，规定数组长度为100，函数名为madd（），数组a，b均定义在函数内，使用gcc编译该程序（使用-g -no-pie -fno-stack-protector选项），

1. 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；
2. 用gdb进行调试，练习下列gdb命令，给出截图；

gdb、kill、quit、break（break \*0x地址 if $rax==401234）、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch

1. 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；
2. 使用单步执行命令及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2022211x\*y的百位和个位。

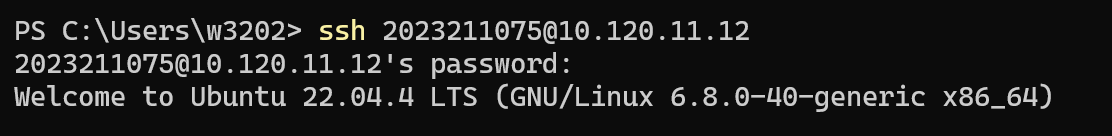
学号2022211999，a[99]+b[99]单步执行前后的参考截图如下（实际命令未显示出）：



**四、实验步骤及实验分析**

## **第一部分：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；**

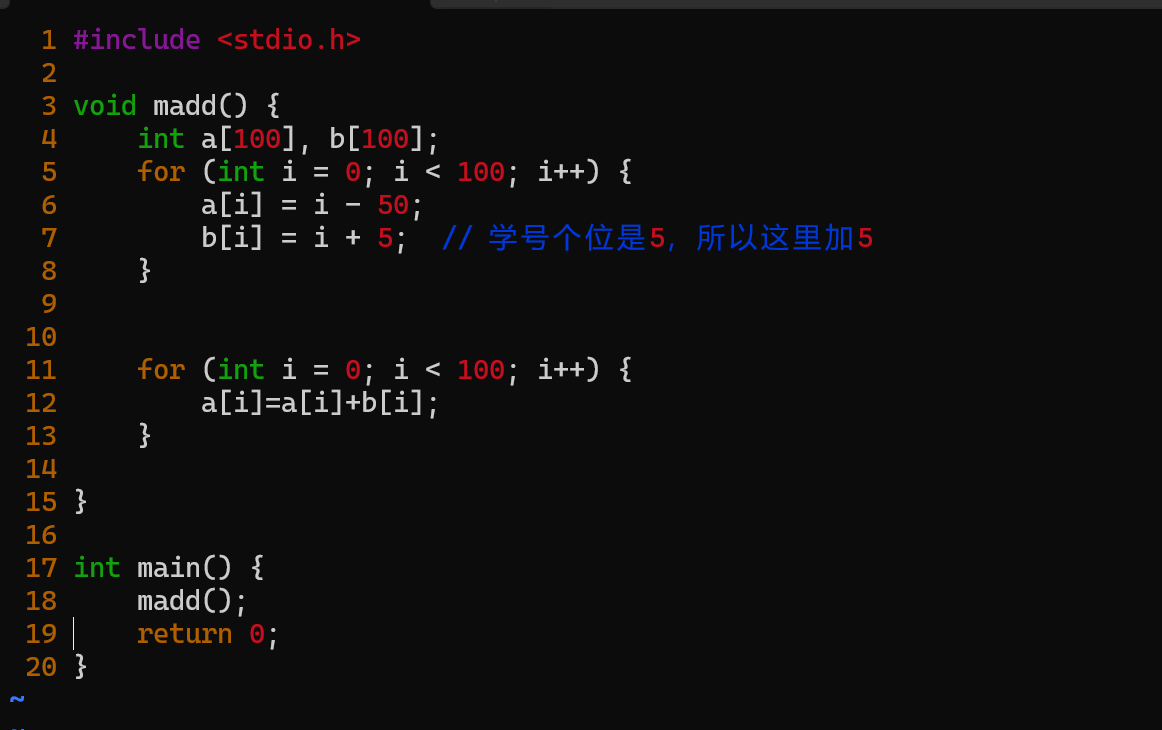
**①步骤一:登录：**

****

**②步骤二：使用vi辑器编写 C 代码**

在终端中使用 vi 编辑器创建一个新的 C 文件

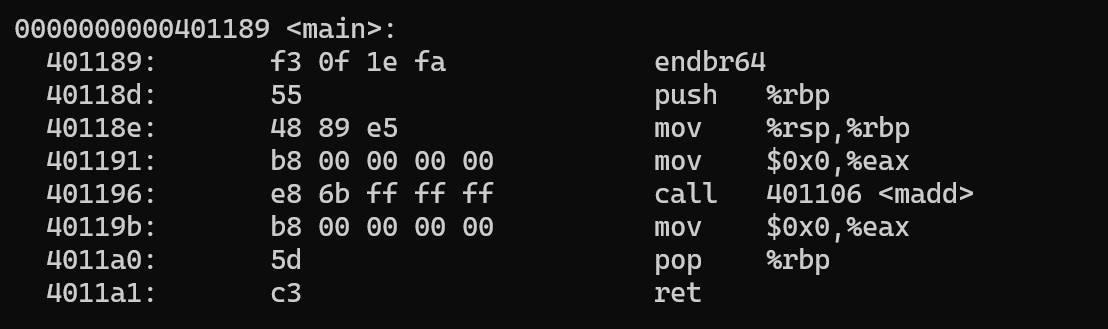
进入 vi 后，按 i 进入插入模式，然后输入以下代码：



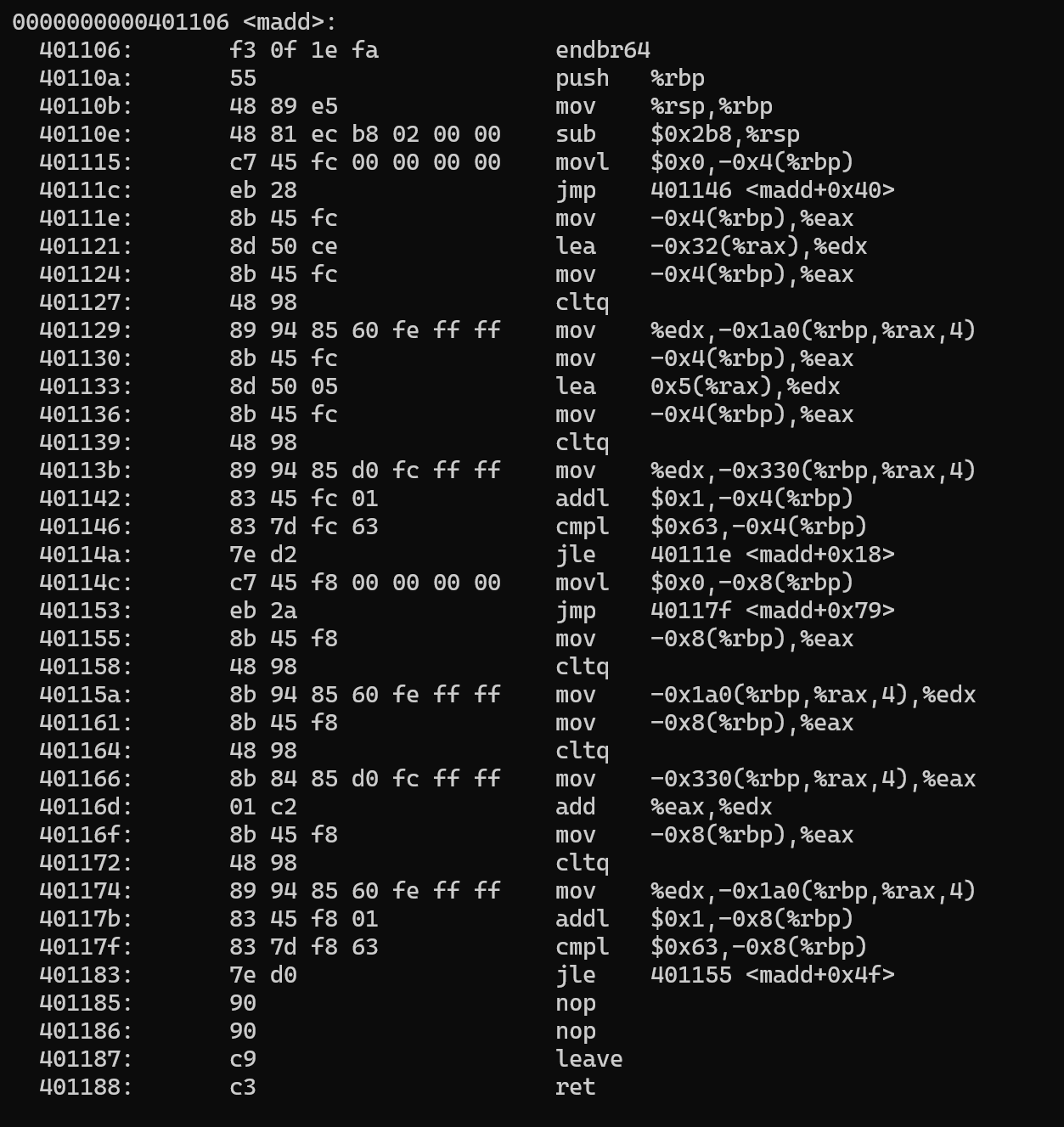
**保存文件并退出**： 按 Esc 键，然后输入 :wq 保存并退出 vi。

**③步骤三：使用 objdump 工具生成汇编代码**

运行 objdump 工具生成可执行文件的汇编代码并找到 madd() 函数的汇编代码部分： 在 madd\_asm.txt 中，查看汇编代码,截图如下：



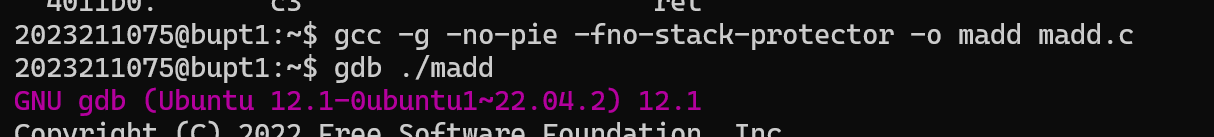
（main部分）



（madd部分）

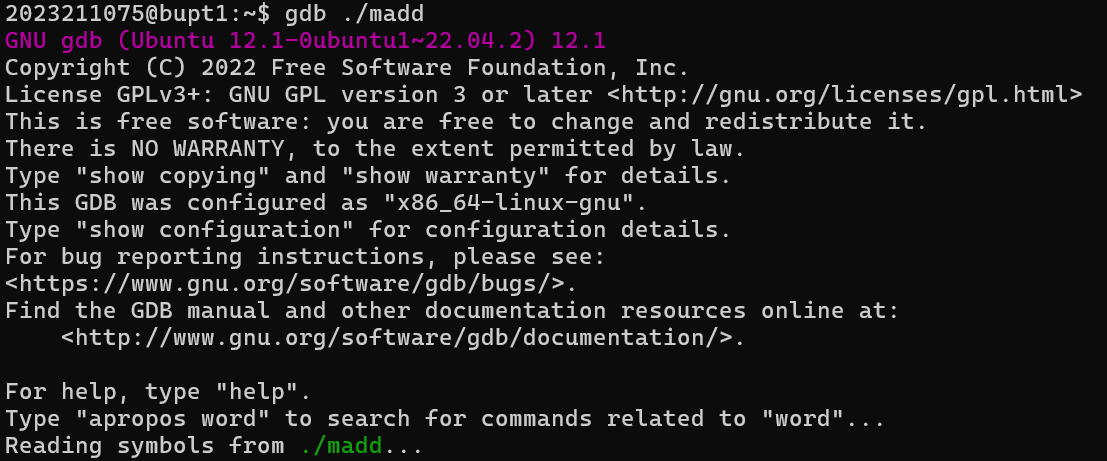
1. **第二部分：用gdb进行调试，练习各种gdb命令，给出截图**

**①步骤一：启动 gdb 并加载可执行文件**

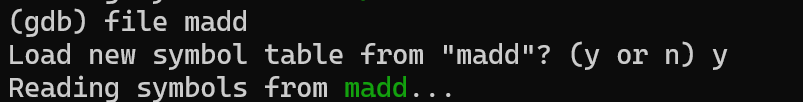


**②步骤二：各种gdb 命令练习**

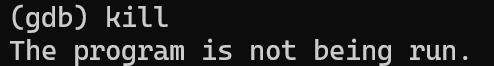
1. gdb



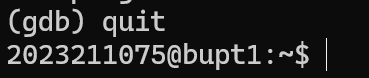
1. file



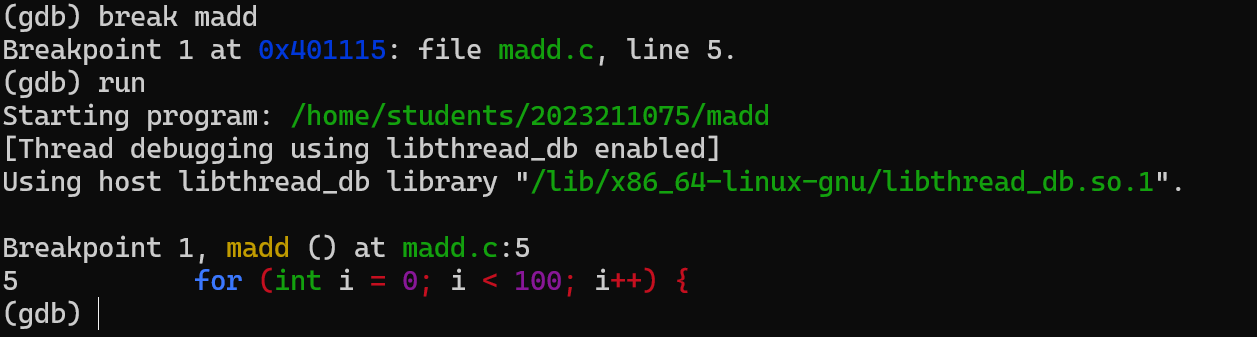
1. kill



1. quit



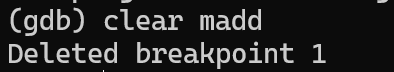
1. break



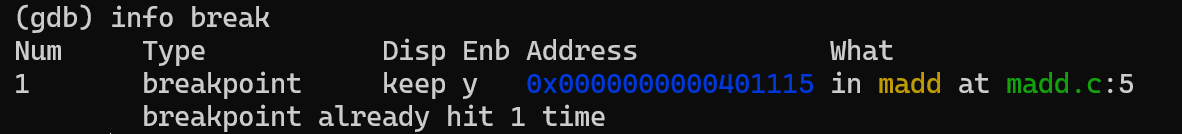
1. delete



1. clear

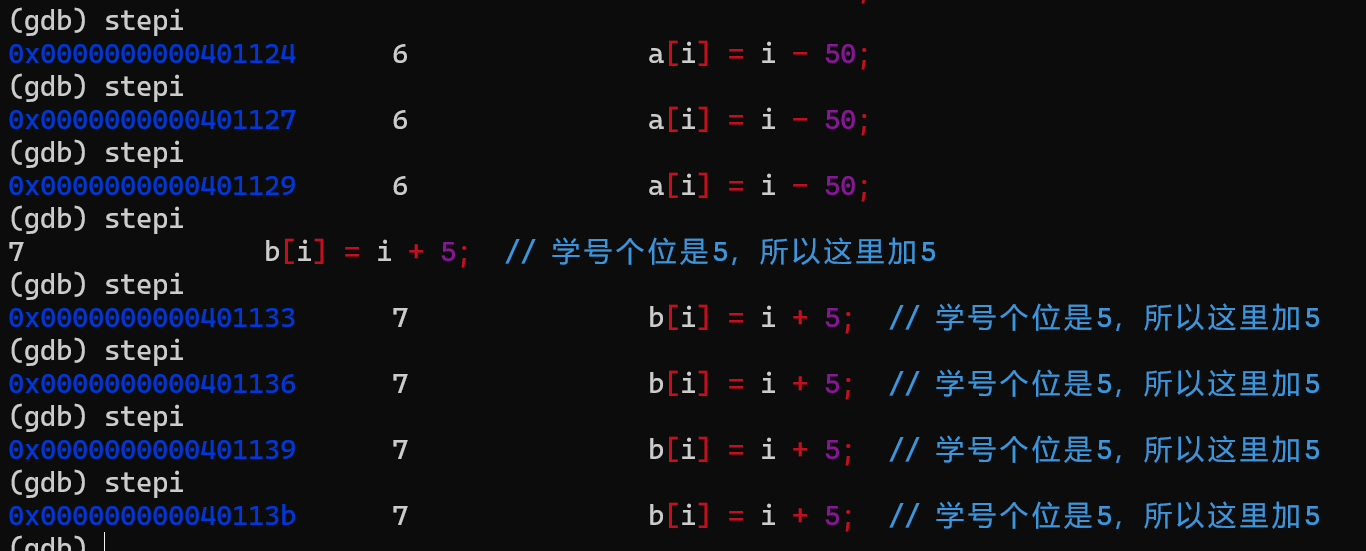


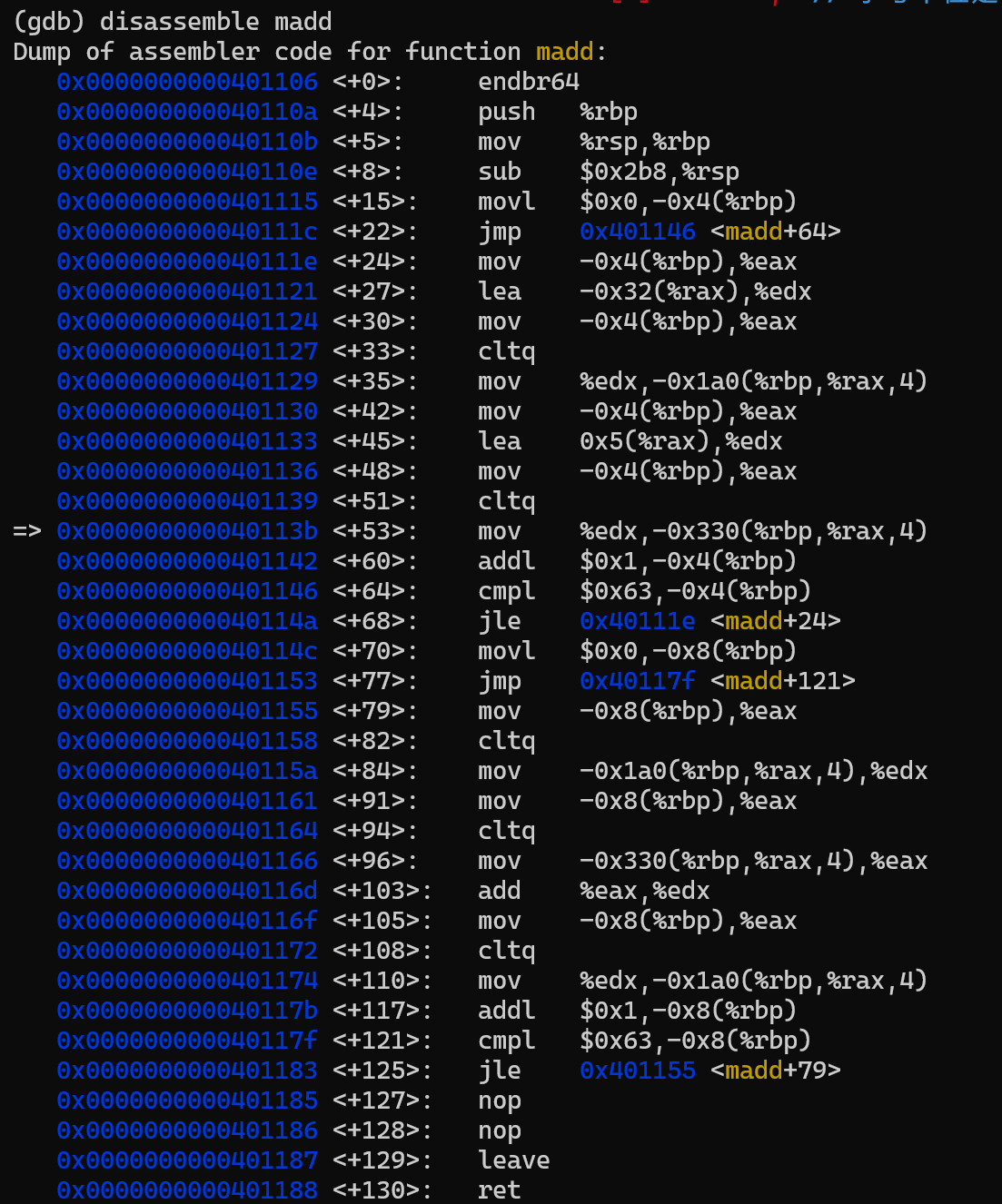
1. info break



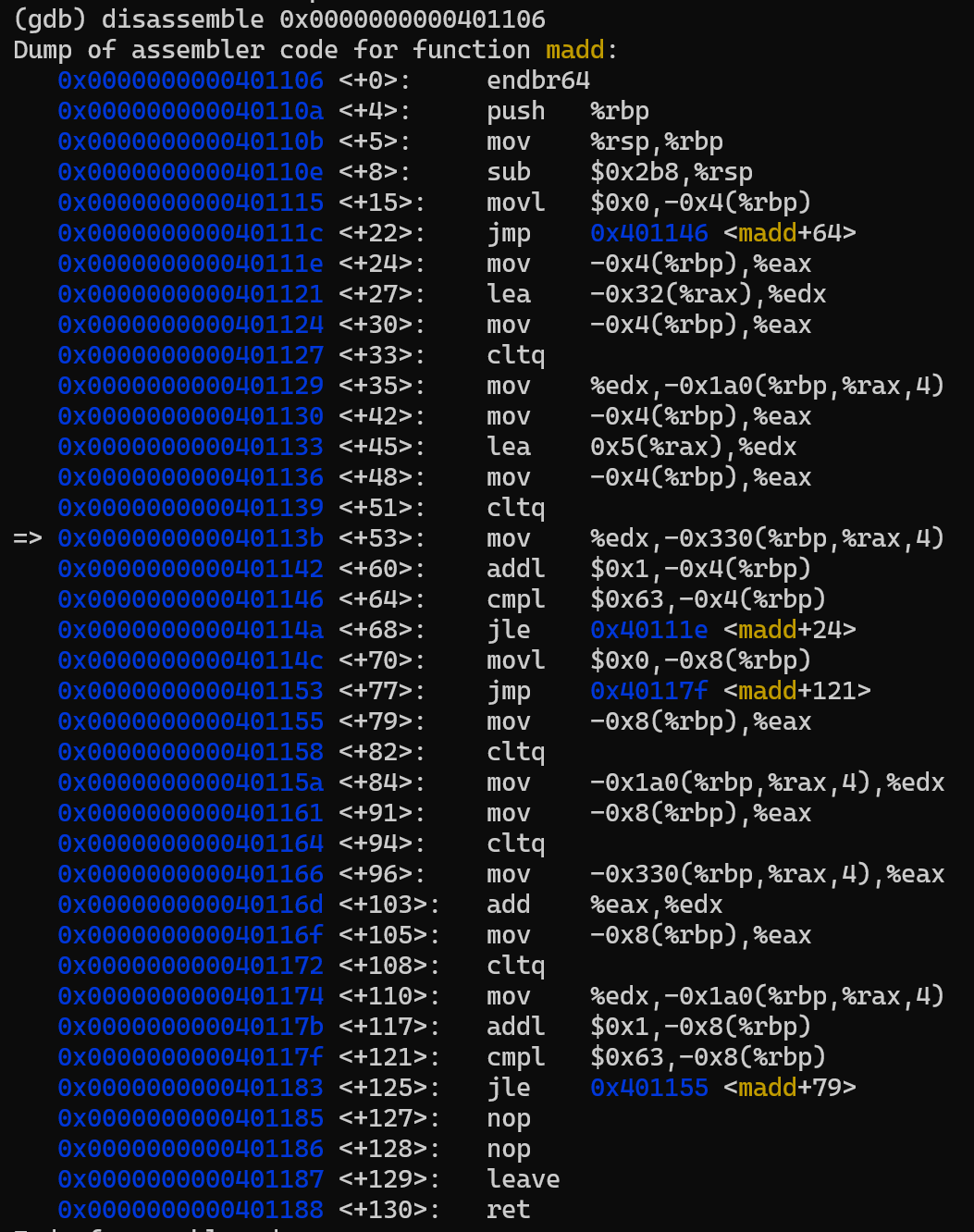
1. run

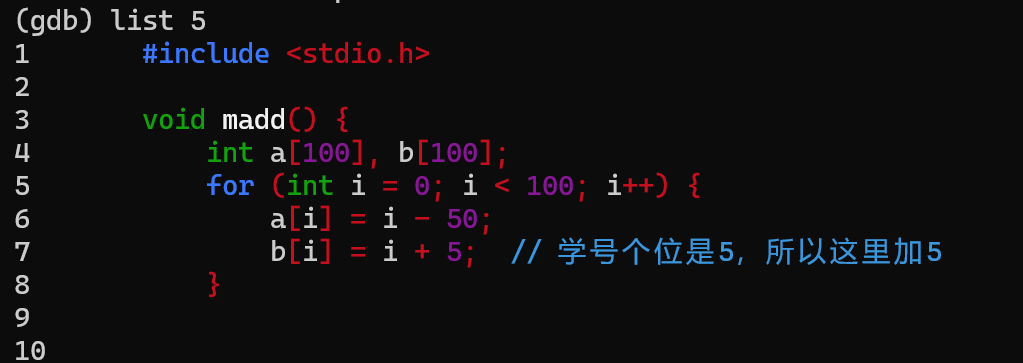
参考break指令下的截图

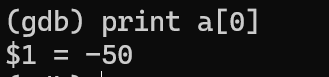
1. nexti
2. stepi
3. disassemble



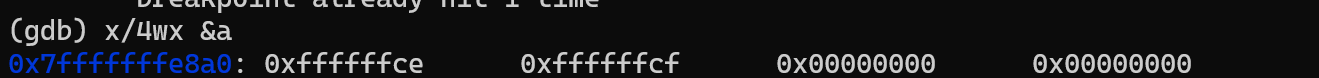
补充：也可以指定要反汇编的地址



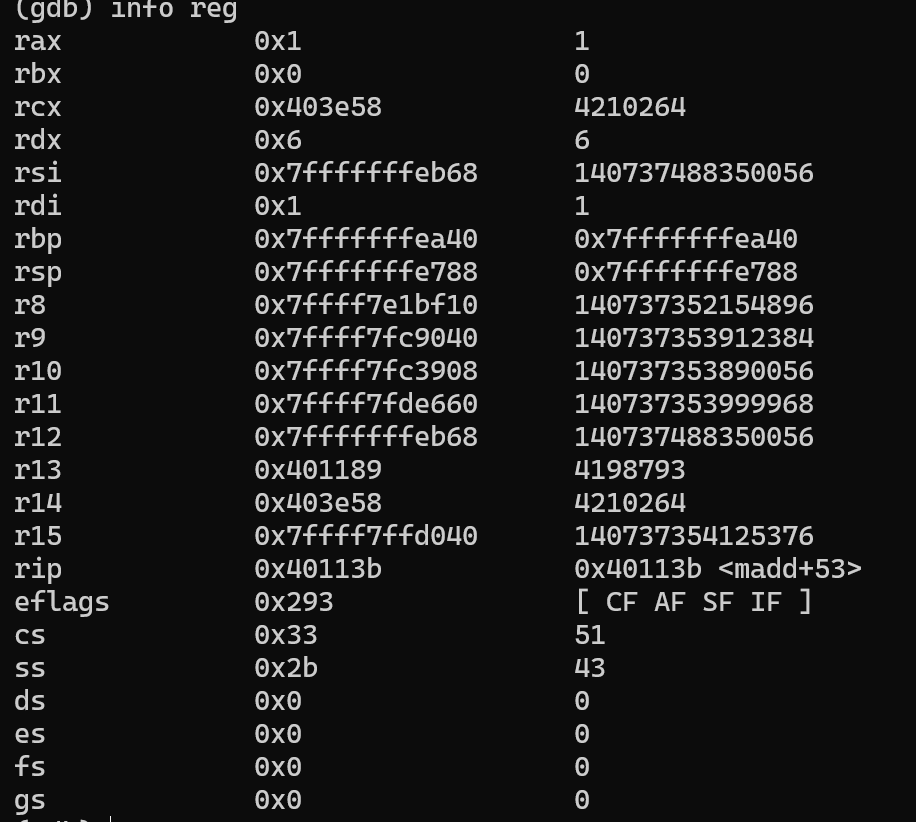
1. list
2. print

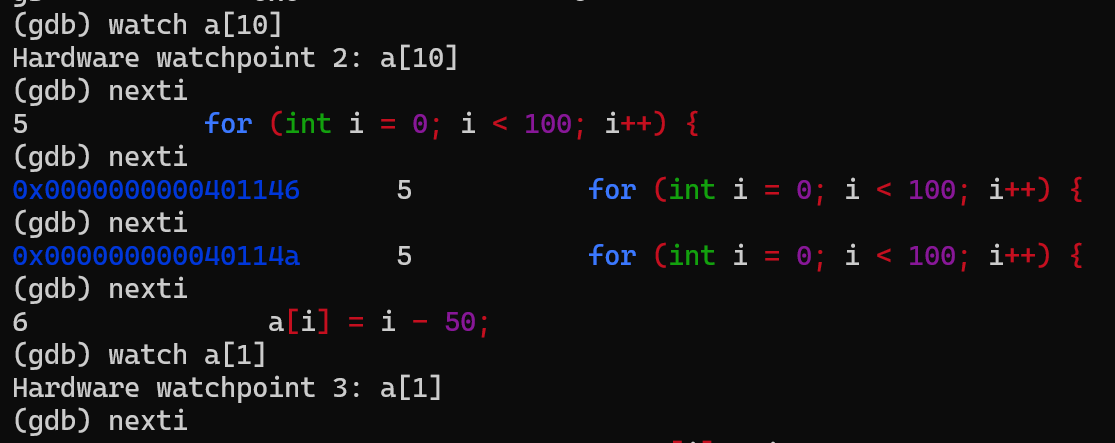


1. x



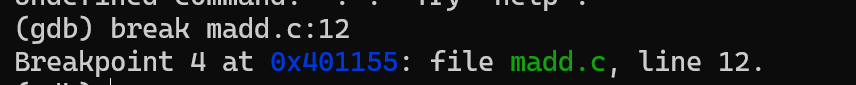
1. info reg



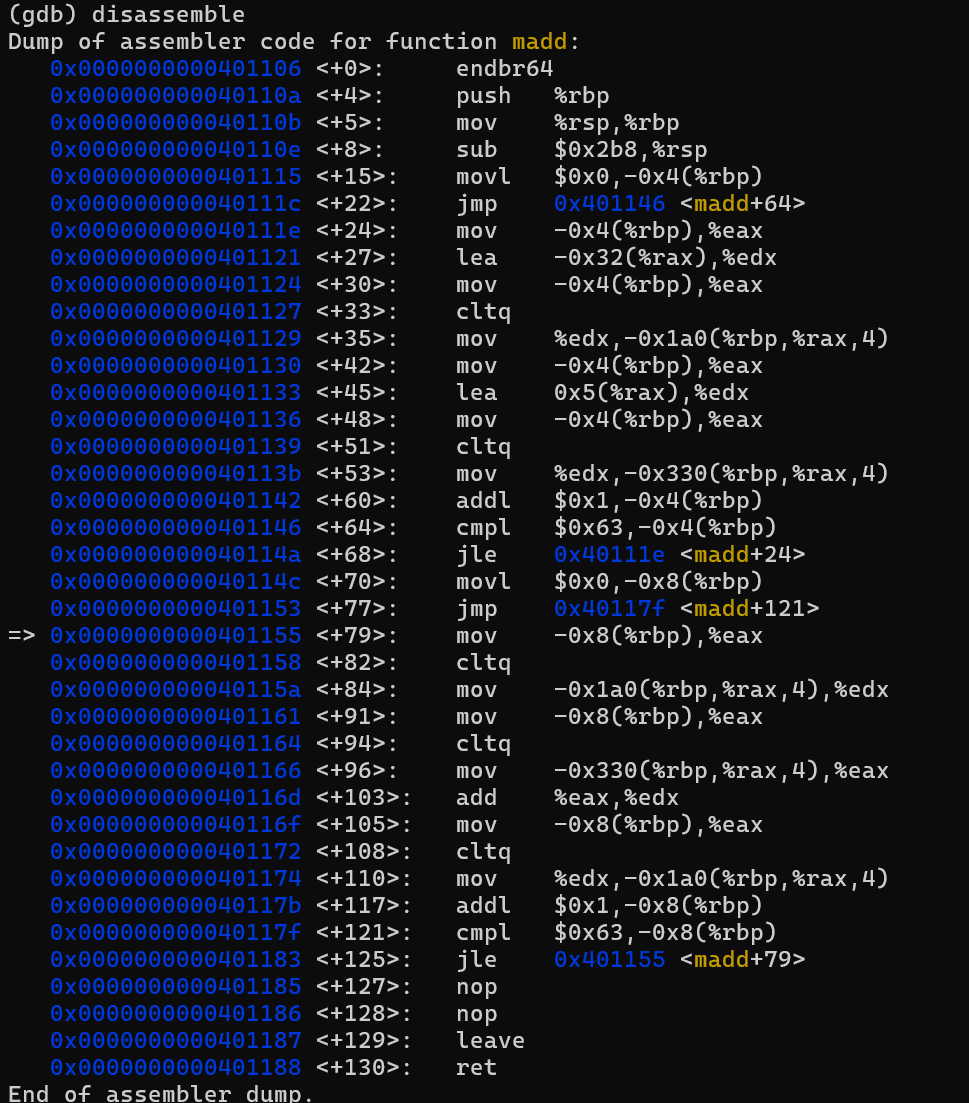
1. watch

**第三部分：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**

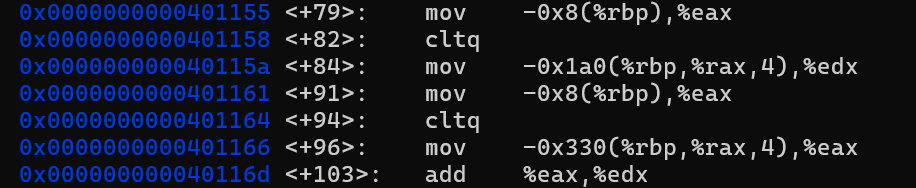
**①步骤一：**设置一个断点在执行 a[i] + b[i] 这行代码的位置（12行）：



②**步骤二**：使用disassemble 来查看该段代码的汇编指令：



**③步骤三**：根据①的断点位置在②中分析a[i]+b[i]对应的汇编指令：



分析这段代码

mov -0x8(%rbp), %eax -这条指令将 a[i]的值加载到 eax寄存器中

mov -0x1a0(%rbp, %rax, 4), %edx -这条指令将 b[i] 的值加载到 edx寄存器中

add %eax, %edx -这条指令将 %eax中的 a[i] 值与 %edx 中的 b[i]值相加，并将结果存储在 %edx 中。

**④步骤四：**得出结论

**- a[i] 的值存储在寄存器 %eax中。**

**- b[i] 的值存储在寄存器 %edx 中。**

**第四部分：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**

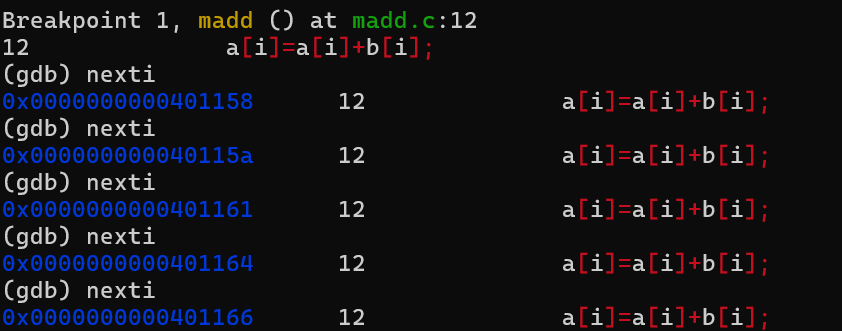
**我学号后三位的百位为0，个位为5，xy=5；**

**①步骤一：**在代码中包含 a[i] + b[i] 的行（第 12行）设置断点，并在 i == 5 时暂停程序

指令**braek madd.c ：12 if i==5**

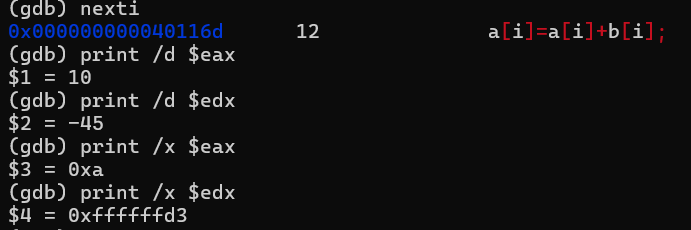
**run**

**②步骤二：**单步执行调试加法指令



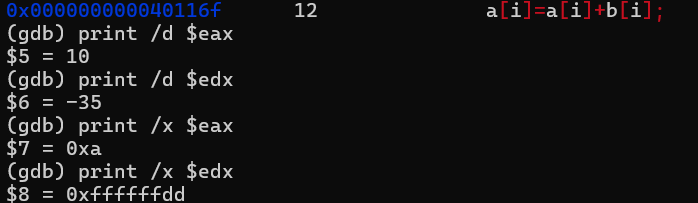
**③步骤三：利用print输出指令执行前后操作数寄存器十进制和十六进制的值**

**执行前**



**利用打印指令 输出查看寄存器十进制和十六进制**

**执行后：**



**④步骤四：与预期对照**

由于 a[5] = - 45，b[5] = 10，

那么加法结果应该是：

a[5] + b[5] = -35寄存在%edx

**综上，符合预期**

**五、总结体会**

通过这次的调试练习，我不仅学会了如何使用 gdb 进行逐步调试和汇编分析，还加深了对程序执行机制、内存操作和寄存器的理解。调试工具不仅仅是找错误的手段，还是帮助你深入了解计算机底层工作原理的利器。

同时，在这个快乐又痛苦的过程中，我也发现了问题并且解决了它们：

问题1:心浮气躁--没理解好汇编代码，在做第三题的时候，我以为只需一步调试就可以达到目的，后来分析了一下代码，发现需要逐步调试。

问题2：急于求成--老是想尽快解决题目，导致很多指令都没记牢，近期一定加以温习。

问题3：粗心大意--最后一个问题的[xy]不是乘积，做完题目才后知后觉，直拍大腿。

最后，感谢老师能提供这次让我熟悉Linux环境和GCC工具链的实验，为徒受益匪浅！